University of California
Santa Barbara

Throughput and Delay on the Packet Switched Internet (A Cross-Disciplinary Approach)

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Daniel Mark Havey

Committee in charge:

Professor Kevin Almeroth, Chair
Fred Baker, Fellow Cisco Systems, Inc.
Professor Elizabeth Belding
Professor Phill Conrad

June 2015

The Dissertation of Daniel Mark Havey is approved.

Fred Baker, Fellow Cisco Systems, Inc.

Professor Elizabeth Belding

Professor Phill Conrad

Professor Kevin Almeroth, Committee Chair

March 2015

Throughput and Delay on the Packet Switched Internet

(A Cross-Disciplinary Approach)

by

Daniel Mark Havey

This dissertation is dedicated to the one true God who created the universe full of scientific wonder for us to explore and enjoy.

Acknowledgements

Funding for this PhD was provided by the University of California Santa Barbara, by Lockheed Martin, by Cisco Systems, Inc, and by Microsoft. This PhD would not have been possible without the support of many people students, professors and many others. I would like to offer thanks to all of my friends from San Bernardino who often said that someday you will be a doctor and I would love to see it. This is the day. We have made it. To all the professors in my undergraduate work who encouraged me along the way and guided me towards research in particular Keith Schubert and David Turner. To the students from my lab at UCSB. To Lara who started after, finished right before and often suffered through tough times with along with me. To David who sat behind me during our perplexing first few years and worked with me on our original TCP experiments that eventually became the starting point of my work. To Rohit, Sathiish and Devdeep for working with David and I during the split TCP experience and to many others too numerous to name here.

A special thanks to my advisor Kevin Almeroth who took me in as a student all of those years ago and stuck with it through tough times and through good. To Elizabeth Belding for asking the tough questions and teaching me a thing or two about humility. To Phill Conrad for his excellent editorial work on the final dissertation and especially to Fred Baker a fine Christian gentleman and "Rock Star" of engineering who taught me about the importance of latency and having a good and cheerful attitude. Fred, you are a man with an amazingly humble servant attitude and a powerful force for the good of the Internet. If I can emulate these two things throughout my career it will be a fine future and worth living.

My mom really deserves extra special thanks for encouraging my love of learning from before I was old enough to talk and to this day. Thank you mom! This is just

a small token repayment for all your love and encouragement. Also a special thanks to Sarah Joy who supported me with prayer and encouragement throughout the final laps of the dissertation process. From providing the logistics for the final defense to organizing the food to helping me wander through the final bureaucracy required to complete the PhD. Thank you for listening to me when I was confused and for praying for me when I was discouraged. I don't think I could have made the final steps without you.

As it says in the Proverbs, "There is gold and abundance of costly stones, but the lips of knowledge are a precious jewel." Mom encouraged my natural love of learning, but, God put that love in place. From the beginning of creation He planned the world including this PhD with all its ins and outs and ups and downs and all of the wonderful people who have encouraged me along the way. He did this so that it would develop me into the person He intends me to be. When I look back at who I was when I started this journey I can see an immense amount of learning and growth both as a researcher and as a person. Now, much like Jesus said nearly 2000 years ago, "It is finished" and it is time to give the honor where it is truly due to our Lord and Savior Jesus Christ. Without Him even knowledge and learning are merely vexation of spirit and a chasing after the wind. I will always remember this and put my knowledge and learning to its proper use to honor and glorify our Lord. Used in this manner knowledge and learning are precious jewels as it says in the Proverb and greatly satisfying to gather, organize and put to use.

Curriculum Vitæ

Daniel Mark Havey

Education

University of California in Santa Barbara (UCSB) – Ph.D. in Computer Science, (Fall 2006 - Summer 2014 estimated)

California State University in San Bernardino (CSUSB) – Bachelor of Arts in Computer Systems, (Fall 2002 - Spring 2006)

Research interests

Emerging markets, mesh networks, Adaptive Bit Rate (ABR) video streaming, network protocols (application, socket, transport, MAC), wired/wireless, packet scheduling

Academic projects

Advanced transport, UCSB (Sept 2013–current)
Networks in emerging markets such as the one in Macha Zambia are often plagued with challenging characteristics such as slow links, high PER, and large RTTs. These characteristics degrade throughput and latency. In this project we improve throughput and latency characteristics using TCP port forwarding to "split" the large RTT into smaller segments. Leveraging the smaller segment RTTs we will also explore the use of CoDel style buffer management techniques to further improve throughput and latency characteristics.

xTCP, Qualcomm (June 2013–Sept 2013)
DASH ABR video streams are sensitive to packet loss and delay. In this project we design a client side only TCP modification that is immune to packet loss and highly robust to delay. This client side kernel modification uses a "lazy" retransmit scheme not reacting immediately to packet loss to offset the DASH ABR video streams sensitivity to packet loss and delay.

Fast Wireless Protocol (FWP), UCSB (March 2012–June 2013)
As wireless speeds increase with 802.11n and 802.11ac the overhead characteristics become more challenging. In this project we investigate the use of variable packet aggregation techniques to reduce overhead and increase throughput. This project is built with the Atheros ath9k open source 802.11n driver for Linux.

Active Packet Scheduling, Cisco (September 2012–June 2012)
The recent increase in multimedia streaming into the home has intensified the need for good resource management at the home access router. In this project we develop network sensing techniques to determine when a packet scheduling profile should be applied. We also implement an adaptive packet scheduler at the IP layer to provide superior management of network resources.

Parallel TCP, UCSB (June 2011–Sept 2011)
Parallel TCP is known to be robust against packet loss and delay, however, it is also known to have two main drawbacks. It is unfair to other flows, and, it is difficult to distribute the data over the parallel streams. In this project we develop a parallel TCP with an application layer fairness mechanism. The fairness mechanism determines the correct share of bandwidth then distributes the data over the parallel streams round robin. Our parallel TCP is fair with single flows as well as robust against packet loss and delay.

Work experience

Research intern, Qualcomm (June 2013–Sept 2013)
Designed and tested xTCP, a TCP variant that is immune to packet loss and therefore very robust against delay. xTCP is a client side kernel modification built on the Linux 3.8.0 kernel. It uses "packet injection" techniques to disguise loss from TCP and a "lazy" retransmit scheme to retrieve lost packets.

Research intern, Cisco Systems (June 2012–Sept 2012)
Performed a detailed study of ABR and progressive video steams characterizing the behavior of each video flow in context with network conditions. Built instrumentation at the video server and the home access router to read TCP service rate, CWND values, and TCP congestion points. Built an adaptive packet scheduling system using this instrumentation that provides improved resource management to the home access network.

Research intern, Aerospace Corporation (June 2010–Sept 2010)
Designed a ground to space emulation system. This ground-space emulation system component functions as a part of the larger Mobility Satellite Emulation System (MSET). It addresses the requirements for ground segment emulation at high speed and fidelity, with various degrees of mobility.

Research intern, Citrix Online, LLC (June 2009–Sept 2009)
This internship was about bandwidth shaping, modeling, and

adaptivity. Created prototype modules for the Go to Meeting (G2M) product. Conducted experiments with the Citrix Online testbed measuring the benefits to quality of customer experience with the prototype modules.

Research intern, Santa Barbara Labs, LLC (May 2008–May 2009)
Conducted satellite network studies in conjunction with Lockheed Martin. Produced white paper deliverables for the Air Force's TSAT Mission Operations System (TMOS) project. Assisted in the design and implementation of the Mobility Satellite Emulation Testbed (MSET). Used this testbed to examine the behavior of mobile IPv6 satellite networks.

Teaching Assistant, UC Santa Barbara (Fall 2006–Fall 2012)
Held weekly classes for 10–20 students to provide additional detail not available in lecture. Held weekly office hours to answer questions and help students prepare for tests. Designed projects and homeworks for the students

Selected publications

Refereed Conferences and Workshops

- Daniel Havey, and Keven Almeroth, "Fast Wireless Protocol: A Network Stack Design for Wireless Transmission", *IFIP Networking*, May 2013
- Daniel Havey, Roman Chertov, and Keven Almeroth, "Receiver Driven Rate Adaptation", *Multimedia Systems (MMSys)*, February 2011
- Roman Chertov, Daniel Havey, and Kevin Almeroth, "MSET: A Mobility Satellite Emulation Testbed", INFOCOM, March 2010
- Daniel Havey, Roman Chertov, and Keven Almeroth, "Wired Wireless Broadcast Emulation", *Wireless Network Measurements (WiNMee)*, June 2009
- Daniel Havey, Elliot Barlas, Roman Chertov, Kevin Almeroth, and Elizabeth Belding, "A Satellite Mobility Model for QUALNET Network Simulations", *MILCOM*, November 2008
- David Turner and Daniel Havey, "Controlling Spam through Lightweight Currency", *Hawaii International Conference on Computer Sciences*, January 2004

Abstract

Throughput and Delay on the Packet Switched Internet

(A Cross-Disciplinary Approach)

by

Daniel Mark Havey

The Internet has become a vital and essential part of modern everyday life. Services delivered by the Internet are used by people across the planet every moment of every day of the year. The Internet has proven a positive force for good improving the lives of billions of people worldwide. The power of the Internet to deliver this positive good to humanity relies on its ability to deliver life improving services. In my doctorate work culminating in this dissertation I have striven to sustain and increase the Internet's ability to deliver these services and to have a positive good effect upon humanity.

The overarching purpose of this dissertation is to improve the Internet's ability to deliver life improving services. I have further divided this purpose into two goals. To improve the ability of applications operating in challenging network conditions to gain their fair share of the bandwidth resources and to reduce the delay with which these services are delivered. Every service delivered by the Internet consists of Internet objects that are delivered through communication paths across the Internet. The delivery of these objects is defined by the two characteristics; Throughput and delay. Throughput determines how much of an object can be delivered over a period of time and delay determines how long it takes to deliver an object.

These two characteristics determine the Internet's ability to deliver objects across communication paths. Improving these two characteristics (bandwidth and delay) increase the ability of the Internet to deliver objects and thus improve the Internet's

capability to deliver life improving services. To accomplish this goal I present projects along three areas of effort. These three areas of effort are: (1) Increase the ability of applications operating in challenging conditions to achieve their fair share of bandwidth. (2) Synthesize knowledge required to address the effort to reduce delay. (3) Develop protocols that reduce delay encountered in the communications paths of the Internet.

In this dissertation I present projects along these three areas of effort that accomplish the two goals (increase bandwidth and reduce delay) to achieve the purpose of improving the Internet's ability to deliver essential and life improving services. These projects and their organization into areas of effort, goals and purpose are my contributions to the networking sciences.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Overview

The Internet has grown in importance and scale over the past few decades and has become a part of the daily lives of billions of people worldwide. Services provided by the Internet are in use twenty four hours and seven days a week on a massive scale. Services provided over the Internet have become a vital life improving part of our world that even effects people who do not use the Internet. Because of the essential life improving nature of the Internet it is important that we keep the Internet functioning at it's maximum capacity now and into the future.

There are two key characteristics that we use to define the health and quality of the Internet: throughput and delay. There are other metrics that measure the quality of the Internet from various perspectives, but, we have chosen these two because they define the nature of how Internet objects are delivered. A flow without enough throughput or with too much delay will be a poor conduit for the delivery of objects that comprise an Internet service. In addition, throughput and delay are the metrics used to define the quality of the transport and Internet Protocol (IP) layers of the network stack

from which the Internet is constructed. The IP layer is a packet switched addressing protocol and the transport layer is usually the Transport Control Protocol is usually (TCP) but can be the User Datagram Protocol (UDP) or some other variant.

That throughput is a key determining factor in the deliver of Internet objects is fairly straightforward. Throughput measures how much data is transmitted over a period of time. With more throughput available more data can be transmitted in the same amount of time. However, latency is a little more subtle. Internet objects (web pages, app data, etc) require a minimum number of Round Trips (RTs) to retrieve because of DNS lookups, TCP opens, SSL/TLS negotiation and HTTP request/response. The time required for an RT is usually expressed as Round Trip Time (RTT) and is the amount of time required for data from the sender to travel accross the network to the receiver and for the ACK to travel back from the receiver to the sender. An Internet object may contain many sub-objects each requiring their own minimum number of RTTs. An object requiring 5 RTTs to retrieve will take about 1 Second to retrieve at 200 ms RTT regardless of the throughput. This time adds up quickly with the number of objects. As an example the current version of cnn.com contains over 40 objects. Because of this minimum RTT time encountered there is little benifit in adding bandwidth after a certain point. Reducing the path latency however, continues to reduce the RTT and dramatically reduces object retrieval time.

The packet switched Internet is an end-to-end communication system. The application layer (at the sending host) is home to a wide diversity of applications that use the TCP/IP stack. The transport layer is responsible for flow control and the IP layer addressing and routing. The IP layer hands the data off to a link layer protocol for actual transmission. Like the application layer the link layer is home to a variety of protocols and technologies that are used for actual transmission of data; Ethernet, Fixed Broadband, WiFi, Mobile Broadband and others. The data travels through any

number of intermediary hosts: Network Address Translators (NAT), proxies, switches, routers and others. Usually middle boxes only have the IP and link layers, though some proxies implement split TCP breaking the end-to-end principle. After traversing the intermediary path the data reaches the receiving host and is received by the link layer, up to the IP layer, through the transport and to the receiving application. The return path is the reverse of this.

Any service delivered by the Internet goes through this communication process in order to deliver its objects. The Internet path used by a service to deliver its objects determines the throughput and delay encountered. The throughput is equal to the smallest throughput of any device in the path. The delay is determined by the slowest device in the path. The throughput and delay characteristics of an Internet path are determined by the slowest device in the path with the least throughput. Typically these are the same device but not always. In order to improve the use of throughput and reduce the impact of delay I first needed to find the slowest link in the typical Internet path and the one with the lowest throughput.

I have identified two key areas that present a good opportunity for improving the throughput and delay characteristics of flows on the Internet from now and into the future. These two areas are in the **last mile wireless link** between the end host and the **access link** between the router and the ISP equipment. The last mile wireless link often experiences transmission loss. The link layer implements retransmission schemes in order to cope with this loss. However, in rural and third world areas the transmission characteristics are often so challenging that the retransmission system is overwhelmed causing loss of throughput. In addition, in saturated metropolitan areas there are often so many collisions that the retransmission scheme is overcome causing loss of throughput.

The second area is in the access link. The access link is a resource that is shared

by all of the flows that are using this access link. However, the access link is controlled by an entity (the ISP) that is external to the end hosts even though the flow control is implemented by the end hosts. This leads to a problem called the tragedy of the commons defined in Hardin's work as a dilemma *"...arising from the situation in which multiple individuals, acting independently and rationally consulting their own self-interest, will ultimately deplete a shared limited resource even when it is clear that it is not in anyone's long-term interest for this to happen."* [1]

The problem is that protocols and applications share network resources (particularly the access link). An application or protocol can reduce its latency by controlling its sending rate. However, an individual application or protocol has little incentive to do so because if even one application or protocol sharing the resource does not behave in this manner then all will share its fate. The flow that has reduced its throughput in order to reduce latency will not only still experience latency caused by the misbehaving flow, it will also have reduced its throughput for nothing. In fact the typical TCP transport behavior is aggressive and will try to grab as much throughput as possible for itself. If all flows are doing this then the finite resource of throughput will be divided about equally among them. However, the latency will be uncontrolled. This arrangement not only provides no incentive for a flow to try to reduce latency, it actually punishes those that do with reduced throughput.

The solution to this problem is to implement queue control to reduce latency at the access link and reduce the queue size of all flows so that each gets its fair share of the throughput without adding unnecessarily to the latency. This queue sizing protocol needs to be implemented beyond the control of any single flow so that it can be enforced upon each and every flow that is using the shared resource. This is a difficult problem since the queue size for each flow is determined individually for each flow according to

[1] http://www.sciencemag.org/content/162/3859/1243.full

its unique throughput and delay characteristics. Some flows do not require the use of their full fair share of the throughput resource at the access link. These flows should be allowed to do this and separated from the other more aggressive flows so that they are not interfered with in their good behavior.

So we have two problems. One is that in the last mile challenging wireless conditions often prevent a flow from reaching its full fair share of the throughput and the other is that flows competing for resources at the access link often cause excessive latency with their aggressive behavior. I address the first problem in this dissertation in two ways: (1) By increasing and controlling the aggression of flows that are operating in the challenging conditions, (2) By causing the flow control system to ignore losses that have been caused by the link layer retransmission scheme being overwhelmed. I address the second problem of shared resources at the access link by applying queuing management discipline to all flows using the shared resource.

I developed protocols at both the MAC and the Session layer designed to improve the throughput characteristics of individual flows. These protocols achieved the goals that I set out to accomplish increasing the throughput in a significant and fair manner. The Receiver Driven Rate Adaptation (RDRA) protocol from Chapter 2 and the Fast Wireless Protocol from Chapter 3 are robust against loss and latency providing approximately twice the throughput in challenging conditions. The Active Sense Queue Management (ASQM) protocol and the Bandwidth Delay Product (BDP) protocol from Chapter 5 and Chapter 6 provide latency reduction in challenging conditions where queue management is difficult due to rate changes and/or large RTT flows.

1.2 Thesis Statement

The Internet has become a vital part of the everyday lives of billions of people planetwide. throughput and delay are two inherent characteristics for every flow on the Internet. Increasing the share of throughput for challenged flows and adjusting the network queue size close to the bandwidth delay product increases the value of the Internet by enhancing its capability to provide life improving services.

1.3 Dissertation Organization

The dissertation organization is shown in Figure 1.1 organized into three areas of contribution to the networking sciences. The first category is increasing throughput in challenging network conditions. The second category is a distillation and amalgamation of knowledge from the networking sciences which are required to understand the throughput delay tradeoffs caused by queue sizing. The third category is designed to reduce latency without sacrificing throughput. The first contribution category is represented in this dissertation by two algorithms; Receiver Driven Rate Adaptation (RDRA) and the Fast Wireless Protocol (FWP). The second contribution area is a collection of essential knowledge adapted from various disciplines in the networking sciences. The third category of contribution is represented by the Active Sense Queue Management (ASQM) and the Bandwidth Delay Product (BDP) algorithms.

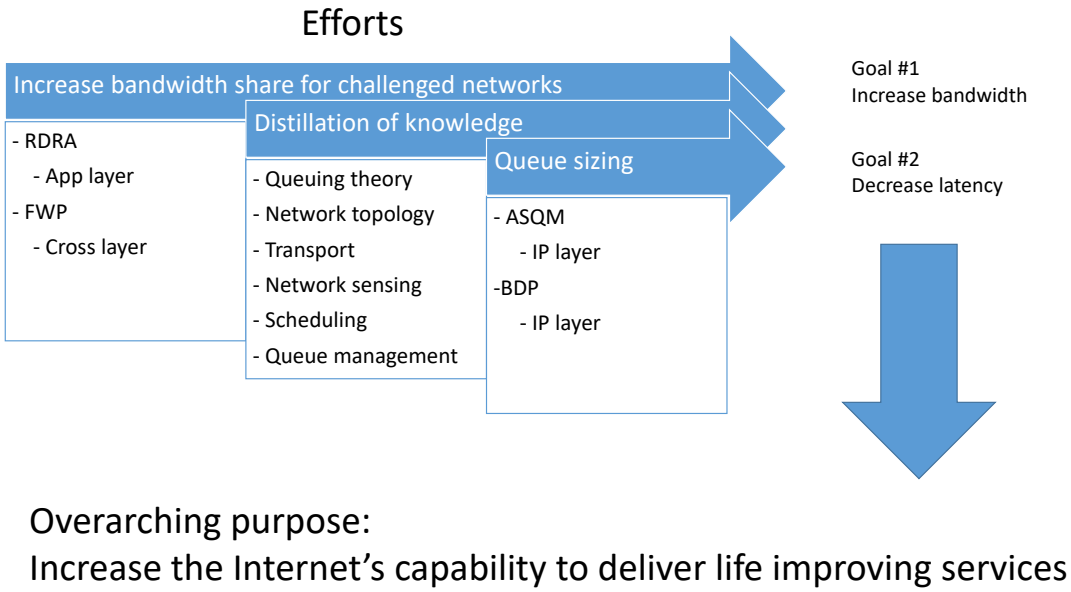


Figure 1.1: Chapter organization by topic

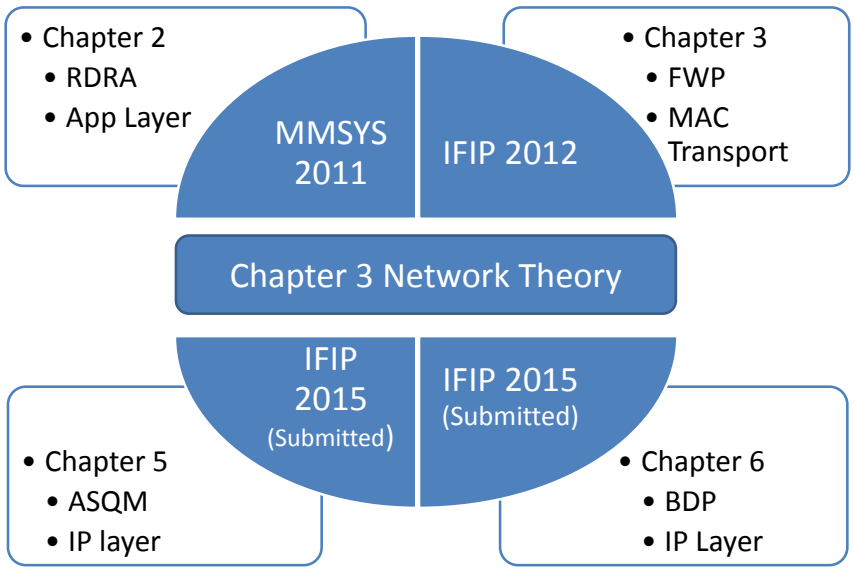


Figure 1.2: Chapter organization by protocol and layer

Figure 1.2 shows the the five chapters organized into the three categories of contribution outlined in Figure 1.1. Chapter 2 and chapter 3 are throughput increasing protocols that mitigate throughput loss caused by excessive packet loss. Chapter 4 is collation of knowledge which details the necessary components adapted from the varied network sciences that are necessary to address the queue sizing problem. Chapter 5 and chapter 6 describe queue management solutions that mitigate the loss of throughput and increased latency caused by improper queue sizing.

In Chapter 2, I describe application layer multi-streaming techniques that use parallel TCP in order to increase throughput in challenging networks. Parallel TCP is more robust against packet loss than single streaming TCP or even retrieving multiple objects at the same time (pipling). This is because parallel TCP distributes the data retrieval across multiple streams decreasing the amount of loss caused by packet loss to

each stream. Parallel TCP is unfair because it uses more TCP streams than other flows and thus captures an larger share of the available throughput. My Receiver Driven Rate Adaptation (RDRA) algorithm distributes the data across multiple TCP streams and manages the fairness problem using a novel fairness mechanism that calculates RDRA's fair share of the available throughput (equivalent to a single TCP) and restricts the aggregate flow for all of the streams to the calculated fair share. RDRA is both robust against loss as well as fair.

Chapter 3 describes FWP a cross layer mechanism that uses 802.11 wireless frame sequence numbers to separate wireless loss from congestion loss. FWP mitigates throughput loss caused by inappropriate backoff from non-congestion related losses. FWP injects filler packets as placeholders in order to hide the wireless losses from TCP. FWP holds the data in a reorder buffer at the session layer until the missing data can be replaced before delivering the data to the application layer. FWP increases throughput in challenging wireless conditions by eliminating inappropriate backoff in TCP. In Chapter 4, I describe areas of knowledge from the networking sciences that are necessary to address the queue sizing problem.

In Chapter 5, I describe my Active Sense Queue Management (ASQM) algorithm. ASQM uses a novel sensory mechanism that injects sense packets into the portion of the path that needs to be controlled (defined by network topology). ASQM uses this information in order to control the queue size across an entire link. ASQM has the ability to control queue size even when the problem moves around throughout the link according to the network characteristics of the access link. This is important because the queue sizing problem often shifts into portions of the link that are difficult to control with queue management techniques. ASQM has the novel ability to control these queues indirectly providing excellent queue management regardless of where in the link the queuing problem has shifted.

Chapter 6 describes my Bandwidth Delay Protocol (BDP) Algorithm. BDP uses a combination of active and passive sensory mechanisms to calculate what the RTT of each flow would be if there wasn't any queuing. BDP is unique in that it calculates the exact queue size tailored to each flow according to it's bandwidth delay product. BDP then uses a novel management system which ranges from gentle management necessary for large RTT flows to aggressive management required for small RTT flows. BDP is uniquely adaptable and can manage any flow at any throughput with any RTT.

In this dissertation I address the problem of improving the Internet's capability to deliver life improving services. I do this by three main contributions. 1.) Increase throughput share for applications running in challenging networks. 2.) Collect and adapt knowledge from various areas of the networking sciences necessary to understand the queue sizing problem. 3.) Address the problem of excessive latency and loss of throughput caused by improper queue sizing. I present these contributions in the form of individual works, however, I believe that ultimately a combination of these efforts is the solution. Throughput share needs to be increased for apps in challenging network conditions and queue sizing needs to be controlled. I believe that ultimately the health of the Internet and the good it provides for humanity relies on both of these things, however, the queue sizing techniques I present rely on some fundamental assumptions about the topology of the Internet. In Chapter 7, I describe methods of using these solutions to cope when the underlying assumptions of how the Internet works change. I present this vision for future research directions which allow my queue sizing techniques to be applicable now and into the foreseeable future.

1.4 Contributions

In this dissertation I make several contributions towards the goal of improving the Internet's capacity to deliver life improving services to people worldwide. In this section I describe the individual contributions according to their category described in Section 1.3. My work contains contributions that have impact across several research communities including the end to end community, the bufferbloat community and transport community.

The first category of contribution is increasing throughput for last hop wireless links. My specific contributions include the distribution of data over multiple transport streams, improving the fairness of multi-streaming systems and client side congestion control systems using the HTTP request/response paradigm. In addition, I have made contributions in the aggregation of frames over SISO links as well as in the separation of congestion oriented loss from channel related loss. These specific contributions are detailed in Chapter 2 and Chapter 3. Specifically RDRA increases the throughput of stations operating in challenging conditions by as much as double that of a single stream TCP session and FWP offers perfect separation of channel related versus congestion losses.

The second category of contribution is the collation and adaptation of knowledge from many disciplines of networking science into a coherent amalgamation that is specific to the needs of queue size management. Queue sizing is one of the most missunderstood disciplines in networking science. It is often regarded as simple though it is deceptively complex. Underestimation of the complexity of the problem leads to a belief that the problem has been solved or soon will be solved when there is no real evidence that this is true. The queue sizing problem has been with us for at least 20 years and is with us today. To simply declare the problem solved without

proper study is naive. This is more than just a simple collection of knowledge from the various disciplines including queuing theory, network topology, transport, network sensing, packet scheduling and queue management. Knowledge has been taken from each discipline and distilled and adapted into a coherent and concise collection of information that is necessary and sufficient to understand and address the queue sizing problem on the packet switched Internet. This amalgamation of distilled and adapted information is presented in Chapter 4.

These areas distilled from are queuing theory, network topology, transport, network sensing, packet scheduling and queue management. Queuing theory is necessary because it provides the underlying equations that describe the size of the queue. Networking topology describes where and when the queue should be controlled, transport describes the end to end flow control that interacts with the network queue and network sensing describes methods of detecting the actual size of the queue. Packet scheduling provides the class based queuing that is necessary in order to separate flows with unique and individual queue sizing needs into their own queues and queue management provides the basic techniques that are needed to control queue size.

The third category of contribution was developed from the second. Queue sizing in order to reduce latency and prevent the loss of throughput. My specific contributions in this category include a highly efficient active network sensory mechanism that incurs very little overhead while achieving a high degree of measurement granularity. ASQM protects the link against latency regardless of the rate and even when queuing occurs below the IP layer. These contributions are described in detail in Chapter 5.

In Chapter 6 I describe in detail further contributions in the third category of reducing latency while preserving throughput. These include a combination of active and passive sensory mechanisms that discover the RTT of a flow without any data in its queues (even while there is data in the queues). A novel algorithm that separates

flows into individual queues and calculates their unique queue size determined by their bandwidth delay product. An adaptive management algorithm that adjusts its aggressiveness according to the duration of the RTT and the amount of excessive queuing. This algorithm is unique in that it does not undersize the queue regardless of the RTT of the flow. Together as a cohesive unit these research efforts accomplish the purpose of this dissertation and improve the effectiveness and efficiency of object delivery over the Internet

Chapter 2

The Receiver Driven Rate Adaptation (RDRA) Algorithm

2.1 Introduction

Wired networks are stable and rarely lose packets except when their queues become full due to congestion. Packets in wireless networks however, are frequently lost due changes in the wireless transmission channel characteristics or collisions. The ambiguity as to the source of loss is problematic for the transport protocol. TCP (the most common transport protocol) is particularly sensitive because it expects that all segment loss is congestion related. Non-congestion related loss due to transmission causes TCP to lose throughput. A great deal of effort has been expended in order to create MAC layer protocols which can hide these losses from the network and transport protocols. However, in spite of the remarkable success of these protocols excessive non-congestion related packet loss can still occur in particularly poor conditions such as those commonly found in third world nations, rural areas and even in crowded metropolitan areas, [71, 45, 18, 26]. In addition, the retransmissions contribute to excessive and highly variable delay across the link.

The problem with packet losses occurring due to non-congestion related sources is

that the ubiquitously deployed TCP transport protocol cannot distinguish the cause of a packet loss and it treats all losses as congestion. This leads to an inappropriate backoff when packet loss is not caused by congestion and a significant loss of throughput. Many alternatives to TCP were investigated as a solution to this problem such as the Datagram Congestion Control Protocol (DCCP) which adds congestion control to UDP, [49]. DCCP allows the Congestion Control (CC) algorithm to be redesigned in order to accommodate non-congestion related packet losses. The advantage of DCCP is that the CC algorithm can be changed independently of the TCP protocol therefore it will not break applications that use the TCP protocol.

Another UDP based protocol is the The Real-time Transport Protocol (RTP), [28, 74]. The RTP protocol has become popular for use in low throughput audio applications such as radio program streaming. The RTP specification contains the RTP Control Protocol (RTCP) which transmits periodic control protocol packets to be used for flow and congestion control. However, RTCP is has become irrelevant because the audio streaming applications where RTP is popular have extremely low throughput requirements and CC is not necessary. Though these solutions are innovative and could have been effective the Internet has converged to the use of TCP for nearly all traffic, [18, 26, 60]. This is likely because Internet Service Providers (ISPs) often drop or delay non-tcp packets.

The convergence on the use of TCP for transport has spurred the development of solutions designed to work withing the TCP framework. Goel et al. reduces sender side buffering in order to reduce latency and increase throughput, [27]. Hsiao et al. used delayed ACKs in order to adapt the flow of packets from the server [38]. These types of solutions work well, however, they require extensive changes to the sender side making them difficult to implement on an individual basis. Parallel TCP is receiver based and simple to implement. Kuschnig et al. demonstrated the benefits of parallel

TCP in video streaming applications, [54, 53, 55, 63, 65]. Parallel TCP style solutions are much easier to implement.

Parallel TCP is an application layer protocol often used to mitigate these effects. Parallel TCP is more robust against large amounts of packet loss and/or delay than single streaming TCP (1 socket connection). However, parallel TCP has a fairness problem. While it is true that parallel TCP increases the fairness share for connections operating with high delay and/or loss, it also decreases fairness for connections operating over more normal conditions. In this Chapter, I present an alternative: the Receiver Driven Rate Adaptation (RDRA) Algorithm is a parallel TCP algorithm that stripes data across n TCP streams. RDRA uses an innovative receiver driven fairness mechanism to manage the throughput share. Using this mechanism RDRA achieves the same fairness as a single stream TCP while maintaining robustness against delay and loss.

2.2 Testbed and Experimental Perimeters

The testbeds that I used for the RDRA experiments are shown in Figure 2.1 and Figure 2.2. I built one testbed in the DETER network security lab. DETER is an Emulab style testbed that I used to construct our RDRA testbed [91]. Emulab is an automated testbed format that allows us to easily run repeatable experiments that are not susceptible to external effects. Unfortunately Emulab has no wireless components so I constructed a second RDRA testbed on the Meraka African Institute for Information and Communications Technology, [44].

The RDRA testbed at DETER is shown in Figure 2.1. All of the nodes in this testbed are constructed from commodity PCs running the Linux 2.6.x kernel. I induced

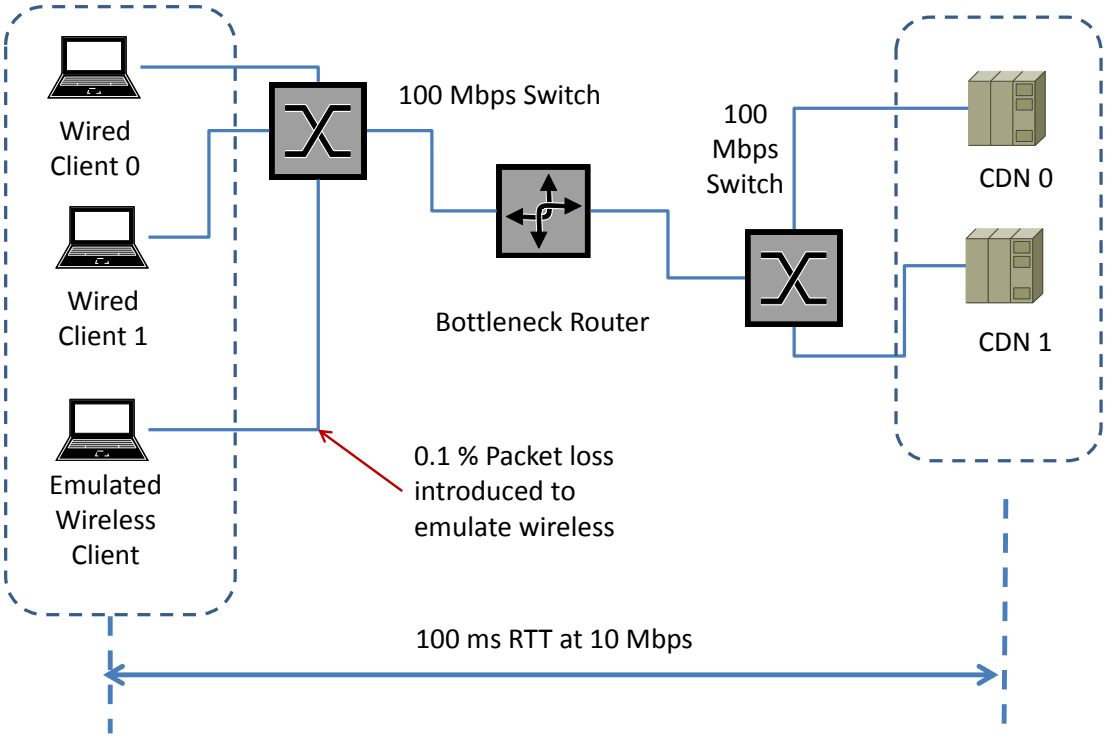


Figure 2.1: DETER Lab Testbed Topology

50 ms delay in each direction at the bottleneck router using netem [1]. To simulate wireless conditions I induced 0.1% packet loss also at the bottleneck router. The DETER testbed is fully automated and suitable for repetitive experiments.

The RDRA testbed at Meraka uses 802.11 abg wireless components. Meraka is a wireless mesh testbed with 49 wireless nodes connected with a 100 Mbps control plane. The wireless nodes are connected with 802.11 abg interfaces spaced about 800 mm apart. Each node is connected to an antenna with 5 dBi gain through a 30 dB attenuator. The path loss between nodes is 60 dB and the radio signal power is reduced so that each node can reach its one hop neighbors but not two hops. Our

[1] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

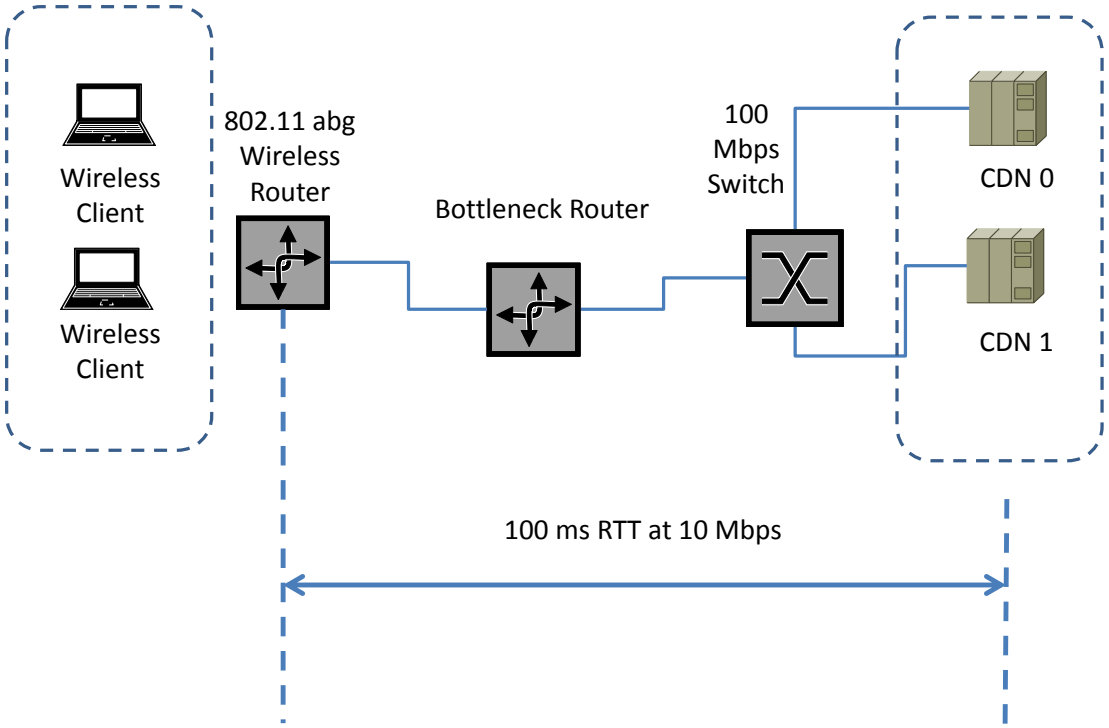


Figure 2.2: Meraka Wireless Testbed Topology

RDRA testbed at Meraka is shown in Figure 2.2. The wired portion of the RDRA testbed is constructed on the control plane and the wireless portion on the wireless data plane. I introduced a delay in each direction of 50 ms at the bottleneck router for a total RTT of 100 ms (plus any delay introduced by the wireless link).

2.3 Introduction to Parallel TCP

Parallel TCP is frequently used in order to improve application performance in terms of throughput and delay. Parallel TCP is commonly used because it is easy to deploy. Application programmers can simply open multiple sockets through their net-

work API. Because of the simplicity of implementation parallel TCP is widely deployed in browsers and is being integrated into video streaming applications. However, the simplicity with which parallel TCP can be invoked by application programmers may hide the disadvantages involved in the use of multiple streams.

Much work has been done to address the fairness problem created by parallel TCP. Solutions such as TCP FIT, EMULTCP, and MULTFRC are n adaptive where n is the number of flows [43, 90, 13]. However, there is little utility in increasing the number of flows beyond a certain point. In fact having too many flows can lead to a phenomenon called self-interference. In practice 3 sockets are enough to obtain %90 utilization and 6 sockets will only yield %95 utilization showing the case of diminishing returns by increasing the number of flows further, [2]. Many parallel TCP solutions achieve fairness through the use of a kernel modification, [31, 46, 12, 30, 57, 63]. However, using a kernel modification limits the utility and ease of use that has made parallel TCP popular.

2.3.1 Throughput and Fairness

Parallel TCP is often used by programmers in order to obtain more throughput for their applications. Parallel TCP increases throughput by increasing the aggregate share of the network queue used by the application. The graphs in Figure 2.3 and Figure 2.3 show the robustness of parallel TCP against packet loss and its fairness characteristics. As the number of parallel TCP streams increases the robustness against packet loss increases and the fairness decreases. To test this assertion I conducted experiments in our DETER testbed from Figure 2.1 comparing two flows competing for resources in a bottleneck router. Each wired client downloaded a large (10 GB) file from a CDN. One flow is parallel TCP with an increasing number of streams and the other is a single

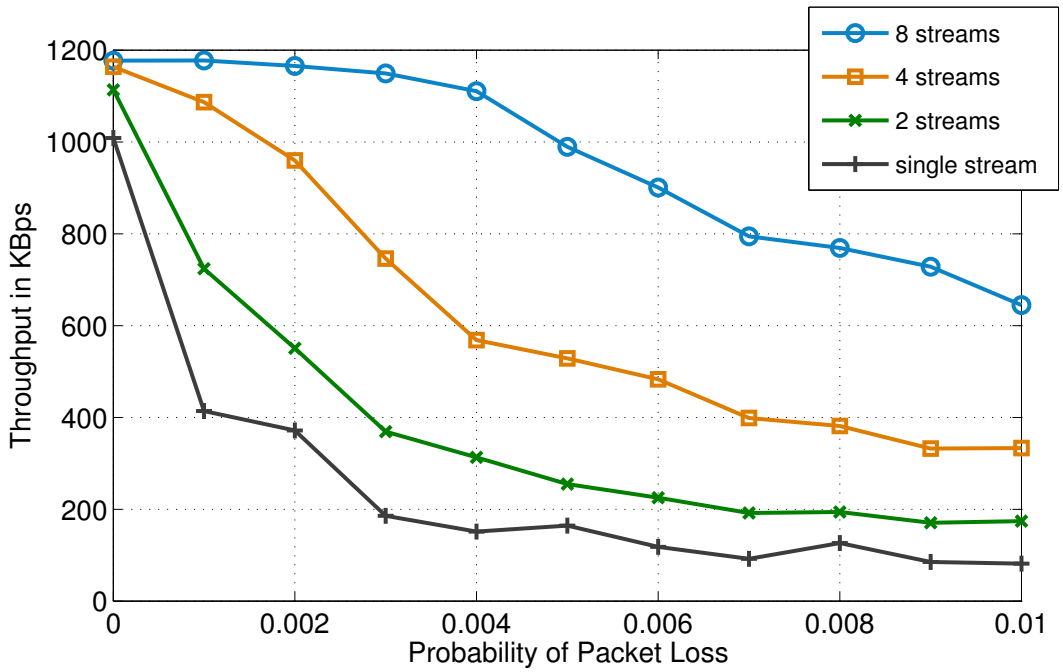


Figure 2.3: Throughput of Multi-Stream Flows

stream TCP.

The theoretical maximum throughput for a TCP flow is given by Padhye's equation, [67].

$$Throughput = \frac{MSS}{RTT\sqrt{\frac{2p}{3}} + RTO(3\sqrt{\frac{3p}{8}})p(1+32p^2)} \tag{2.1}$$

MSS is the Maximum Segment Size, RTT is the Round Trip Time, p is the packet loss, and RTO is the Round Trip timeOut. With a standard MSS of 1500 Bytes and discounting the effects of packet loss a flow with an RTT of 100 ms could develop about 1.5 Mbps. In practice the nominal throughput is less because of packet header overhead and packet loss.

The throughput results of this experiment are shown in Figure 2.3. With zero induced packet loss the single stream flow develops about 1 Mbps. Here we see the effects of diminishing returns as the number of streams increases as described by Altman et al., [2]. With small amounts of packet loss ($< 0.1\%$) most of the throughput gains are

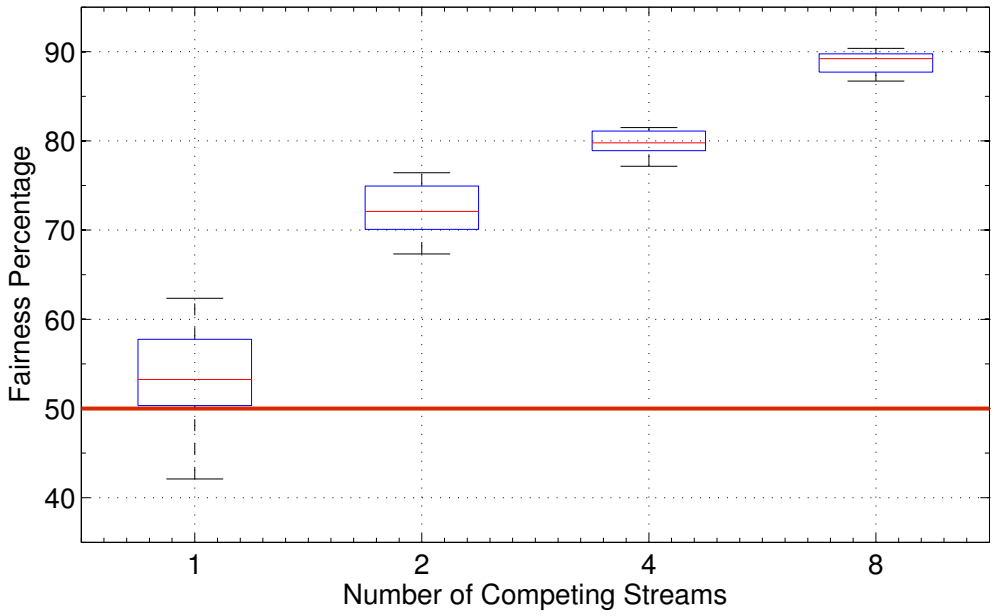


Figure 2.4: Fairness of Multi-Stream Flows

realized by 4 streams and very little is gained by increasing further. As the packet loss increases the single stream flow loses throughput much more quickly than the parallel stream flows. Parallel stream TCP gains robustness as the number of flows increases.

The corresponding graph in Figure 2.4 shows the effects of parallel TCP on fairness as the number of streams increases. The testbed is setup as in Figure 2.3 with two competing flows and the packet loss fixed at 0.1%. There are 10 experimental runs in each whisker with the min and max shown as well as the 10-90th percentile and the mean. The line at 50% indicates perfect fairness where each flow receives exactly the same throughput. As the number of streams increases the fairness decreases.

2.4 Receiver Driven Rate Adaptation (RDRA)

RDRA is a receiver driven parallel TCP that uses rate adaptation in order to maintain fairness while gaining the robustness of parallel TCP. Unlike other parallel TCP systems RDRA is completely receiver based and maintains the simplicity of use for application programmers that has made parallel TCP popular. RDRA requires no sender side or in network changes whatsoever and can be installed on an individual basis. RDRA works by calculating the TCP fair share of the throughput at the receiver and limiting the parallel streams to that rate in order to maintain fairness. RDRA is not an n (number of streams) adaptive algorithm. RDRA uses a fixed number of streams (8) and limits the amount of data requested in order to maintain fairness. RDRA is not an equation based TCP either. RDRA uses a TCP simulator calculating how much the CWND at the sender should be in order to maintain fairness and then only requesting that much data at a time.

2.4.1 Round Trip Time and Packet Loss

As shown by Padhye's equation 2.1 TCP throughput is primarly determined by two parameters; RTT and packet loss. Unfortunately RTT is difficult to determine for reasons that I will revisit in 5. I tried to methods; using the TCP RTT calculation algorithm and using a direct sensing method. I found that both methods produce unsatisfactory results as demonstrated by our series of experiments the results of which are highlighted in Figure 2.5 and Figure 2.6.

This series of experiments shows the results from using a TCP clock that sends a dataless packet (64 bytes) from the receiver to the sender and counts the time elapsed until the returning ACK. The experiments were conducted in our DETER testbed with 100 ms RTT and no induced packet loss. I found that under load the TCP

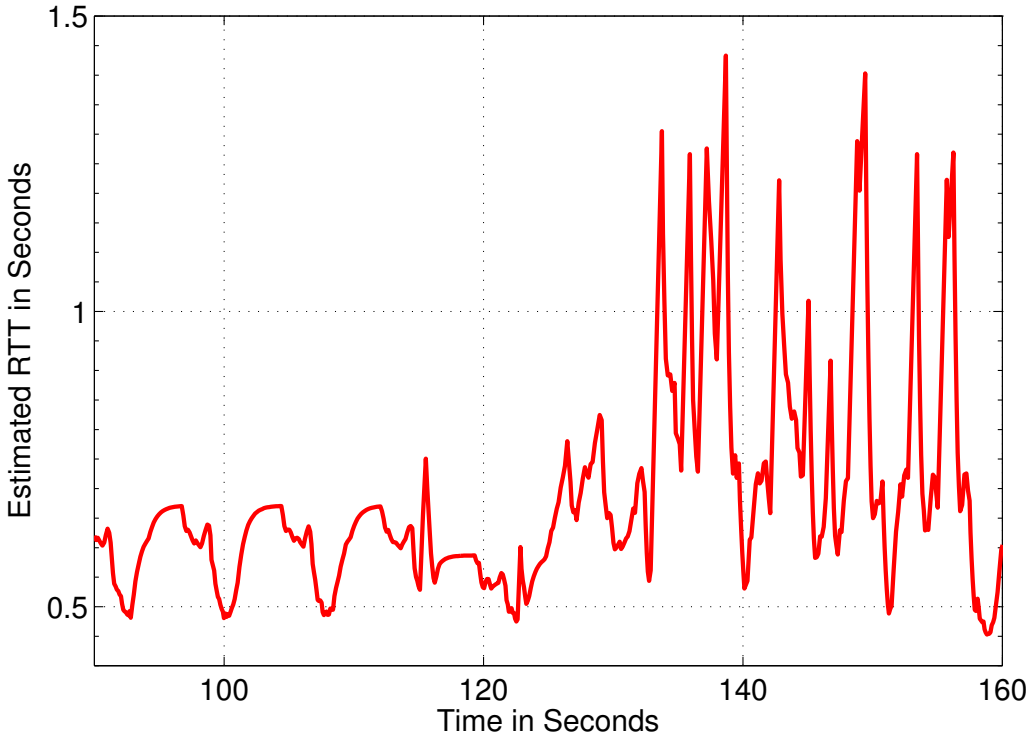


Figure 2.5: RTT Estimate with TCP Clock

clock mechanism produces completely unreliable results. For the sake of space I do not include graphs from the TCP RTT calculation method, however, they were just as unreliable. I used paced TCP in order to control the loading of the queue in the bottleneck router. Figure 2.5 shows the results from a typical experimental run. For the first 120 seconds I paced the TCP to load the bottleneck to about %50 utilization after that I allowed the TCP to run free as shown in Figure 2.6. At no time during any of the experiments did the TCP clock (or the TCP RTT algorithm) produce results that reflected the actual 100 ms RTT. During the %50 utilization period the RTT results were in excess of 500 ms and during the %100 utilization period the RTT results were often in excess of 1000 ms.

Our receiver based RTT measurements were completely unsuitable for use in cal-

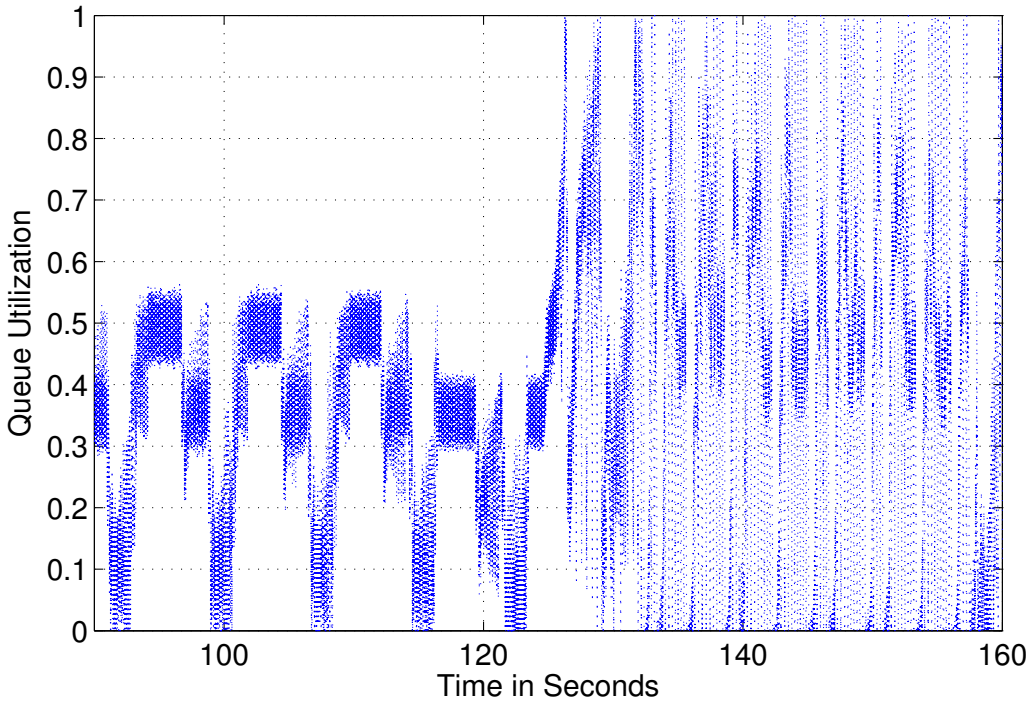


Figure 2.6: Queue Utilization with TCP Clock

culating fair share throughput for RDRA. Fortunately however, our receiver based packet loss measurements were more successful. In order to measure packet loss at the receiver I used an application layer extension to iptables[2] called $netfilterqueue$[3]. Netfilterqueue allows us to separate each flow and track sequence numbers from the TCP headers. Sequence number holes indicate a probable packet loss. In order to determine that our system of receiver side packet loss detection works I conducted experiments using our DETER testbed.

The graph in Figure 2.7 highlights the results from this series of experiments. The experiments consist of a single flow from a CDN to a wired client through the bottleneck router. Packet loss is indicated by the thick vertical lines and queue size is superimposed

[2]http://linux.die.net/man/8/iptables
[3]http://www.netfilter.org

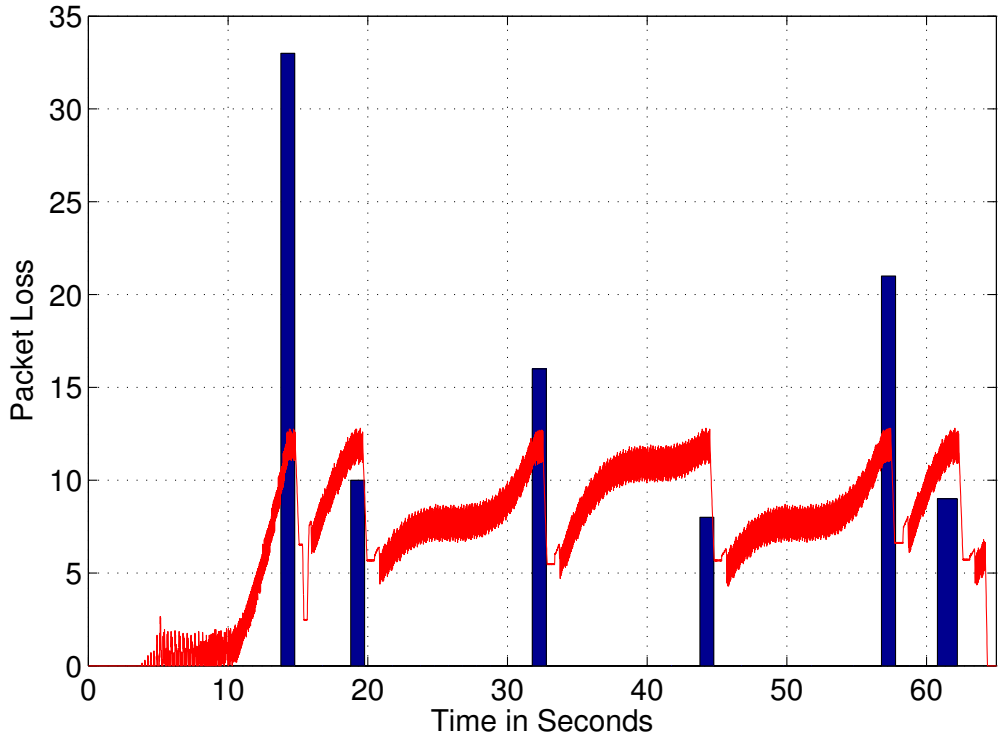


Figure 2.7: Client Side Congestion Detection

on the graph. I can see that the packet loss detection events correspond very well with the congestion backoff events indicated by the queue size. This indicated that I could safely use receiver based packet loss detection in order to determine fair throughput share for RDRA.

2.4.2 RDRA CWND Calculation

Having developed a reliable means for packet loss detection the next step for RDRA was to determine the single stream fair TCP share of the throughput. In order to do this I used the Cubic TCP equation, [29]. The Cubic equation is the default congestion control mechanism for Linux machines and is very popular on the Internet. Cubic requires only packet loss and access to the system clock in order to determine the fair

TCP rate. The Cubic equation is given below:

$$W(t) = C(t - K)^3 + W_{max} \tag{2.2}$$

With W_{max} being the CWND at the time of the last congestion event, t the time elapsed since the last congestion event, and:

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{2.3}$$

With $C = 0.4, \beta = 0.8$ [29]. I chose the Cubic TCP congestion control mechanism for our RDRA experiments because it is both popular and compares on a one to one basis with our Linux testbed. However, much like the pluggable Linux congestion control mechanism any of the TCP congestion control variants can be used [80, 65, 10, 56, 25, 61]. The notable exception to this are the delay based variants of TCP which require an accurate RTT measurement [17, 42, 9, 82, 83, 84, 36, 40].

It is impossible to compare the receiver side CWND calculation with the sender side calculation made by the TCP algorithm because of synchronization issues. However, in Figure 2.10 I present a plot of the receiver side calculation. I can see that the algorithm is functioning as it should and is producing the Cubic curve in values that are commensurate with the throughput being achieved by the sender. This indicated that the CWND Calculation module is working and suitable for use in RDRA.

2.4.3 RDRA System Design

Having developed a good method for packet loss detection and chosen a congestion control algorithm the next step was to build the system architecture. Figure 2.9 shows a block diagram of our RDRA system architecture. RDRA consists of three components

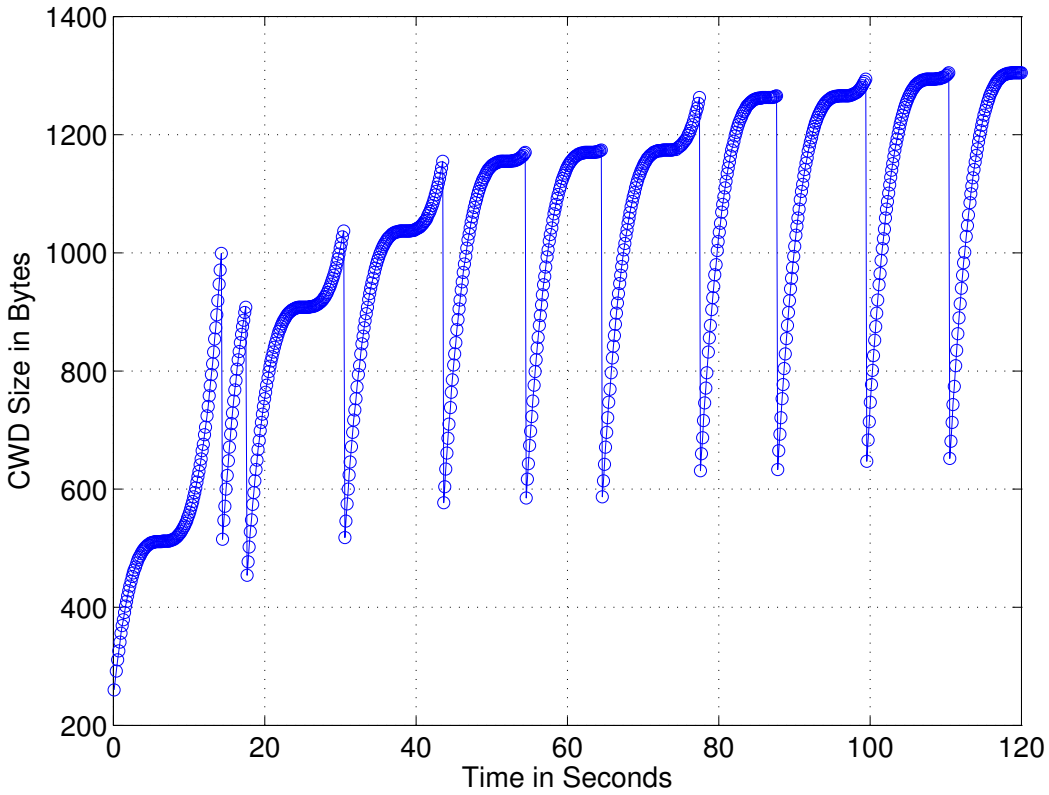


Figure 2.8: Stream Control – Outstanding Requests vs CWND Calculation

and is built in a shim that exists in the session layer of the TCP network stack. As shown in the diagram of Figure 2.9 the TCP stack remains unchanged except for the addition of the RDRA as a shim layer. The client connects to the RDRA system. RDRA stream control splits the flow into 8 streams dividing the data among them as described in Section 2.4.4. The Congestion Detection mechanism detects sequence number holes in order to determine congestion and the CWND calculation mechanism uses a pluggable congestion control mechanism that determines the fair share TCP rate for RDRA. The fair share rate calculation is given to the stream control mechanism and a new fair share TCP rate is divided among the streams.

The RDRA system is both modular and built of modules. Any of the three modules

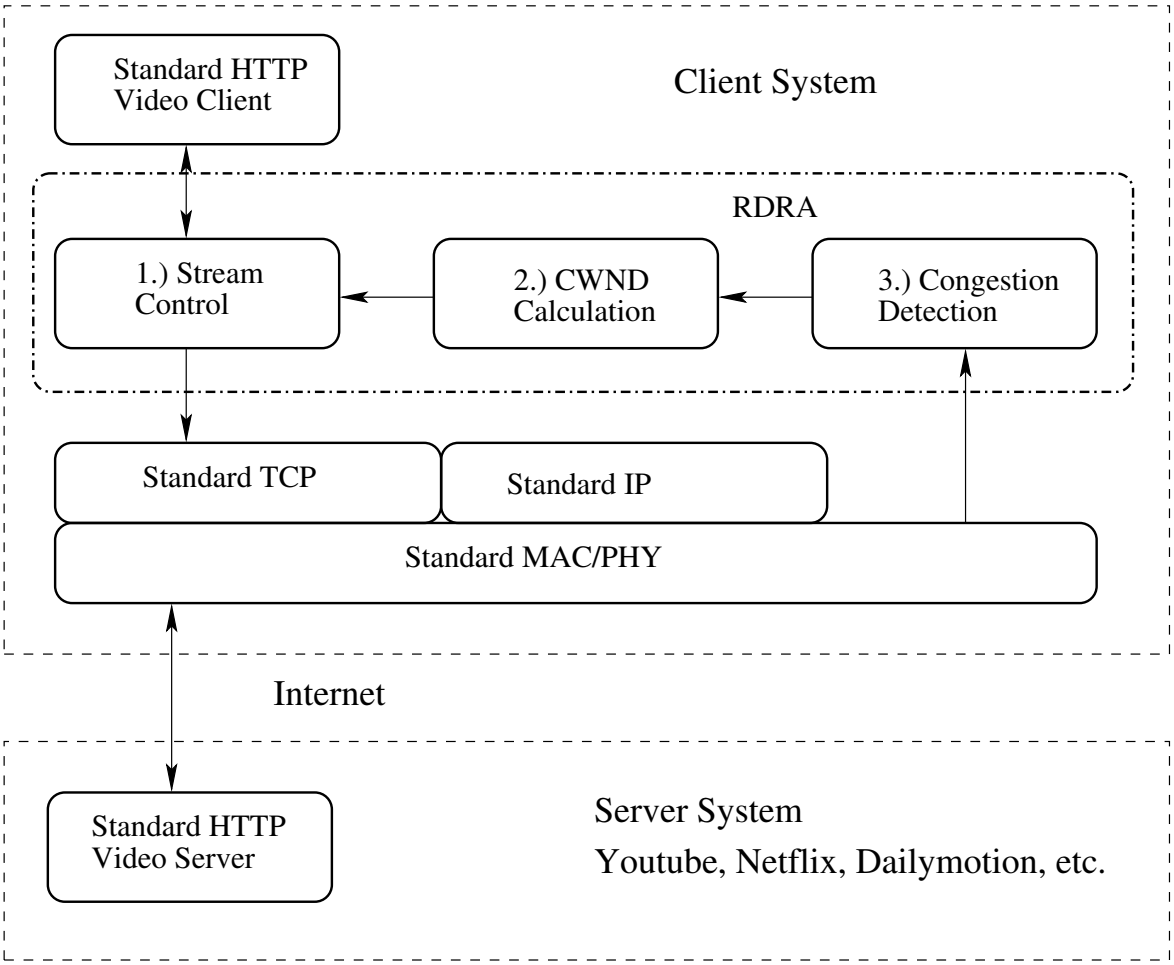


Figure 2.9: RDRA System Architecture

that RDRA is built of can be replaced. Stream Control, CWND Calculation and Congestion Detection can be replaced individually, in pairs or all three and the entire system is a module that fits into the TCP stack at the session layer in the TCP stack. In fact the Congestion Detection module has already been replaced. The original used a teminal based version of wireshark called *tshark* [4]. The current version uses the much more effective *netfilterqueue* iptables extension. This modular construction allows RDRA to be easily upgraded or modified to fit individual user needs.

[4]https://www.wireshark.org/

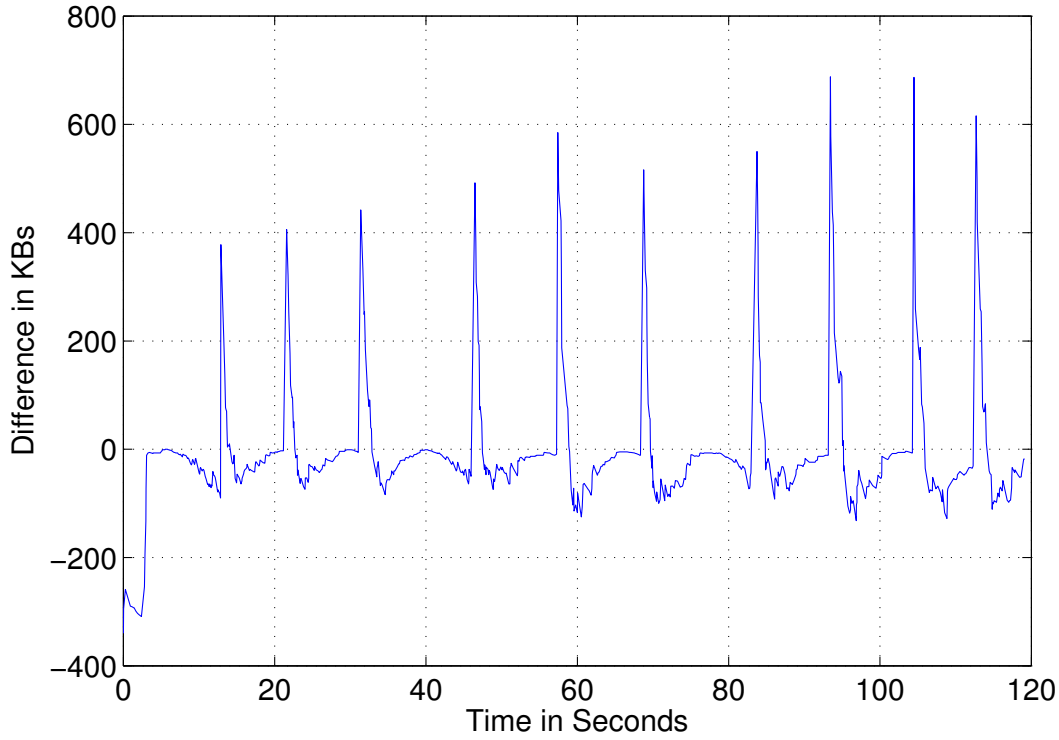


Figure 2.10: Stream Control – Outstanding Requests vs CWND Calculation

2.4.4 RDRA Stream Control

The RDRA Stream Control system is the point where the application connects to RDRA and where RDRA connects to the transport layer of the network stack. I accomplished this by means of a custom built HTTP proxy. Stream control receives an estimate of the fair share TCP rate from the CWND calculation module and attempts to divide the work among 8 TCP streams. Specifically it does this by waiting for an individual stream to complete then if the stream was successful it will check with the CWND calculation module for the latest CWND estimate. The CWND estimate is then used in the following equation in order to calculate the new size for the chunk of

data that should be requested by the stream.

$$chunk_{size} = min\{\frac{CWND_{est} - bytes_{requested}}{nStreams}, 1\ kB\} \tag{2.4}$$

$CWND_{est}$ is the estimated fair congestion window size and $bytes_{requested}$ is the number of bytes outstanding from requests by other streams. $Chunk_{size}$ is in bytes and the smallest chunk allowed is 1 kB. The Stream Control interface uses libcurlmulti to make the parallel TCP requests [5]. If a stream has stalled or failed, then libcurl-multi returns to `Stream Control` after a timeout. `Stream Control` then invokes the libcurl-multi interface closing the stalled socket and opening a new one re-requesting the data. The experiment indicated that the Stream Control module is working as expected and is suitable for use in RDRA.

In Figure 2.10 I show a plot of the Stream Control module at work. I plotted the difference between the CWND estimate and the actual amount of outstanding bytes requested. In the plot I see that the difference remains near or slightly below zero then suddenly spikes upward and just as quickly drops back to near zero periodically. This is due to the effect of one chunk being completed then a new chunk (of a different size) being requested. This experiment demonstrated that the Stream Control module keeps the outstanding requests approximately equal to the CWND estimate provided by the CWND Calculation module.

2.4.5 RDRA Experimental Results

Now that the RDRA modules had been constructed and tested individually the next step was to evaluate the system as a whole. I examined the throughput increase achieved by RDRA along with fairness by comparing RDRA to a single stream TCP

[5] http://curl.haxx.se/libcurl/c/libcurl-multi.html

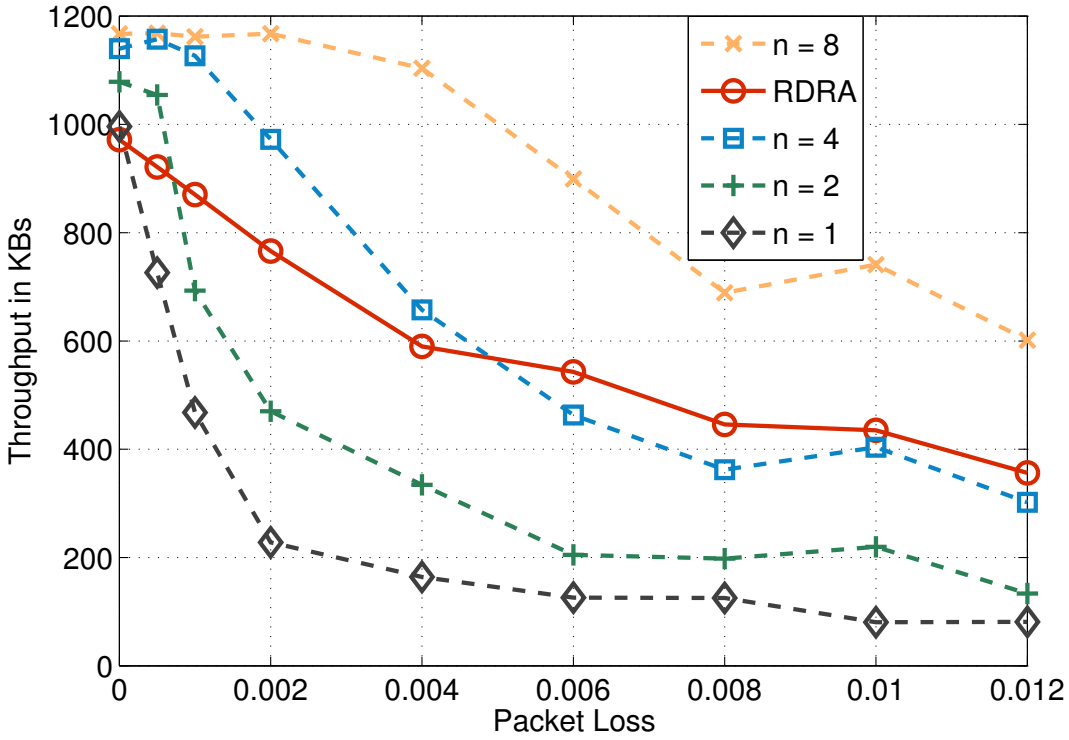


Figure 2.11: RDRA Throughput

flow in our DETER RDRA testbed. In addition, I wanted to look at queue utilization to verify that it is similar to that of a single stream TCP flow. Finally, I wanted to test RDRA against a single stream TCP flow using real wireless hardware in our Meraka testbed.

In Figure 2.11 I highlight the results from a series of experiments in our DETER testbed that shows the throughput characteristics of RDRA against increasing amounts of packet loss. The graph shows that at zero packet loss RDRA is completely fair achieving nearly the same throughput as the single stream TCP. As the packet loss probability increases RDRA retains nearly twice as much throughput as single stream TCP which degrades very quickly. RDRA is more robust against packet loss than single stream TCP. RDRA's throughput line is similar to a 4 stream TCP but less

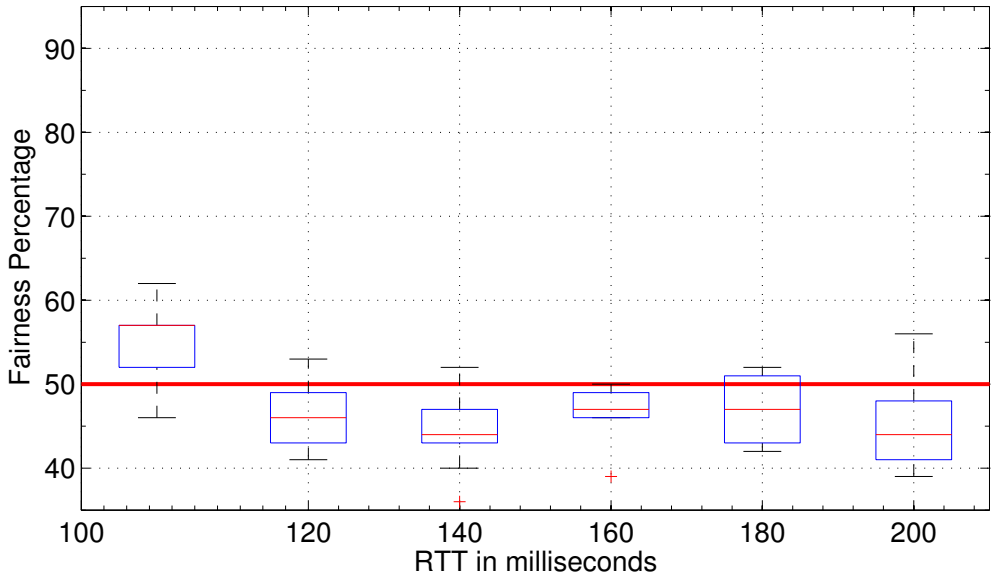


Figure 2.12: RDRA Fairness

than an 8 stream TCP. This is because of the fairness action of RDRA's stream control mechanism restraining RDRA's 8 streams to what a single stream TCP would have achieved without excessive packet loss.

Once I had determined that RDRA has superior throughput characteristics to single stream TCP against packet loss the next step was to determine RDRA's fairness characteristics. In order to accomplish this I conducted experiments in our DETER testbed comparing RDRA at a typical (0.1%) packet loss across a range of RTTs from 100 ms to 200 ms. I highlight some of the results of this series of experiments in the whisker plot shown in Figure 2.12. As in Figure 2.4 the experiment is a competition between two flows where one flow used RDRA and the other used single stream Cubic TCP. The experiments were repeated 10 times and the min/max whiskers displayed along with the 10-90th percentile and the mean and the line at 50% indicating perfect fairness where each flow receives exactly the same throughput.

In Figure 2.4 two single stream Cubic TCPs consistently achieved between 40 and

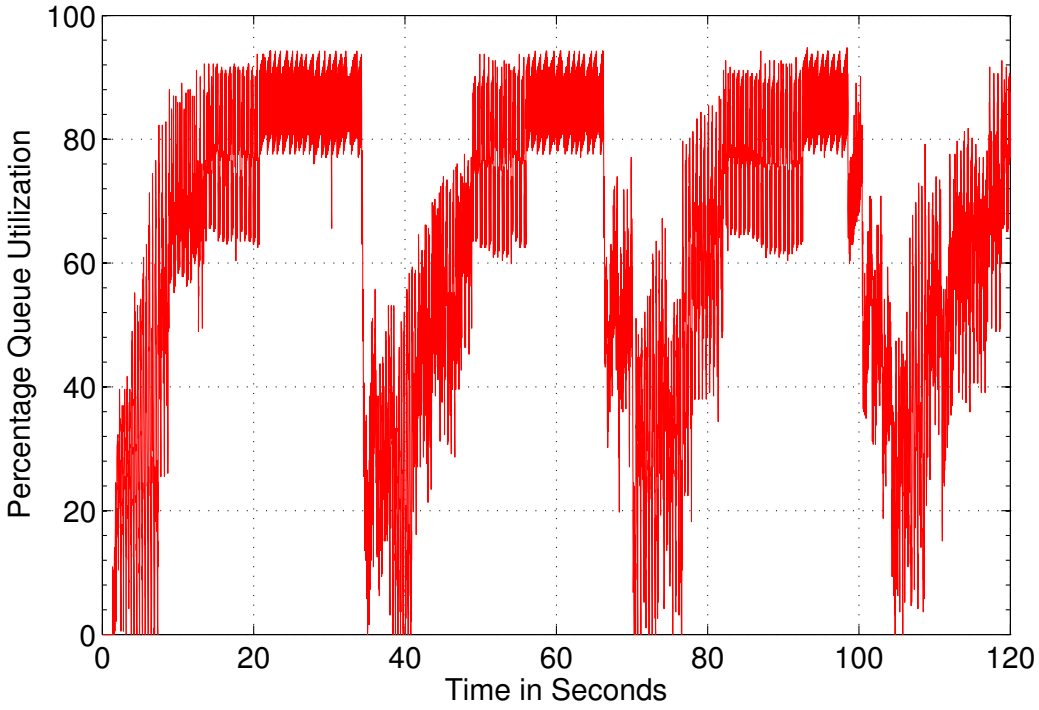


Figure 2.13: RDRA Queue Utilization

60% on our fairness scale. I deem this to be TCP fair behavior. The graph in Figure 2.12 shows that RDRA maintains fairness (with the exception of a few outliers) across the entire range of RTTs. RDRA achieves the benefits of parallel TCP while maintaining fairness across a wide range of RTTs. I did not test RDRA's fairness against increasing packet loss since from the graph in Figure 2.3 that parallel TCP does lose throughput against increasing packet loss, it just doesn't lose as much as single stream TCP.

Having determined that RDRA is both robust against packet loss and fair with single stream TCP the next step is to examine RDRA's queue utilization characteristics. In Figure 2.13 I highlight the results from one of our series of queue utilization experiments performed in our DETER testbed. The RTT is 100 ms and there is no induced packet loss. In Figure 2.16 I show the queue utilization of a single stream TCP

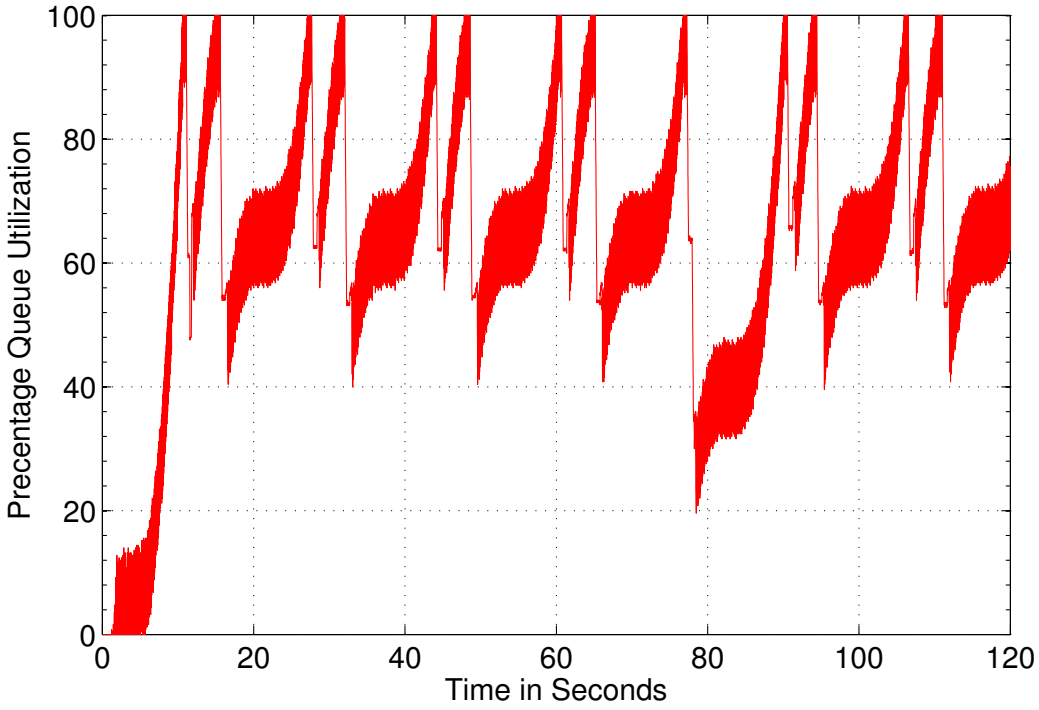


Figure 2.14: TCP Cubic Queue Utilization

for comparison. The graphs show that RDRA maintains a similar queue utilization as single stream Cubic TCP approximately 50-80%. RDRA like all parallel TCP is jittery with respect to queue utilization. This is okay because the reason for having queues is to absorb the jitter. However, in later work described in detail in Section 4 I have found that high queue utilization is not a good thing because it increases latency. This is the reason why I had so much trouble with our RTT measurments described in Figure 2.5 and Figure 2.6. RDRA or any TCP based flow either single streaming or parallel should be used in conjunction with queue sizing algorithms such as those described in Sections 5 and 6.

The final series of RDRA experiments that I conducted used real wireless hardware from our Meraka testbed. In this series of experiments I was trying to determine how much throughput would typically be gained over a single streaming TCP in the face

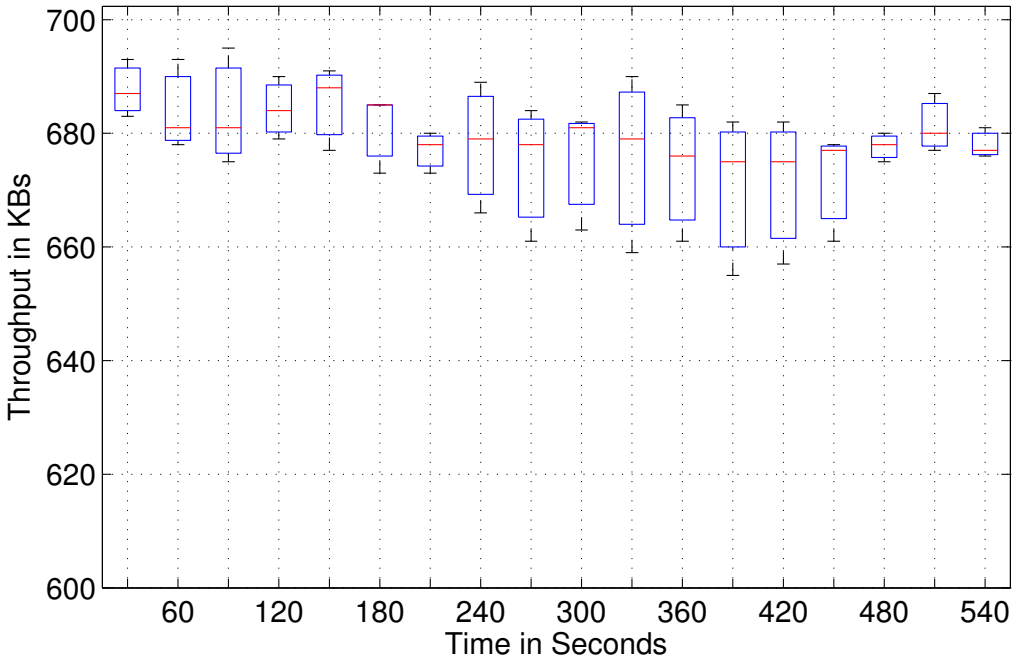


Figure 2.15: RDRA Throughput

of wireless packet loss. I used the testbed topology described in Figure 2.2 with two competing flows one using RDRA and the other using a single streaming Cubic TCP. The graphs in Figure 2.15 and Figure 2.16 highlight typical results from this series of experiments. The y axes of the two graphs are different for the sake of visibility. On the left hand side in Figure 2.15 I show the throughput results from the RDRA flow averaging a little less than 700 Kbps throughout the duration of the experiment. On the right hand side in Figure 2.16 I show the results from the single stream TCP flow averaging a little less than 200 Kbps throughout the duration of the experiment. This approximately 250% increase in throughput was consistent throughout all of the experiments in this series.

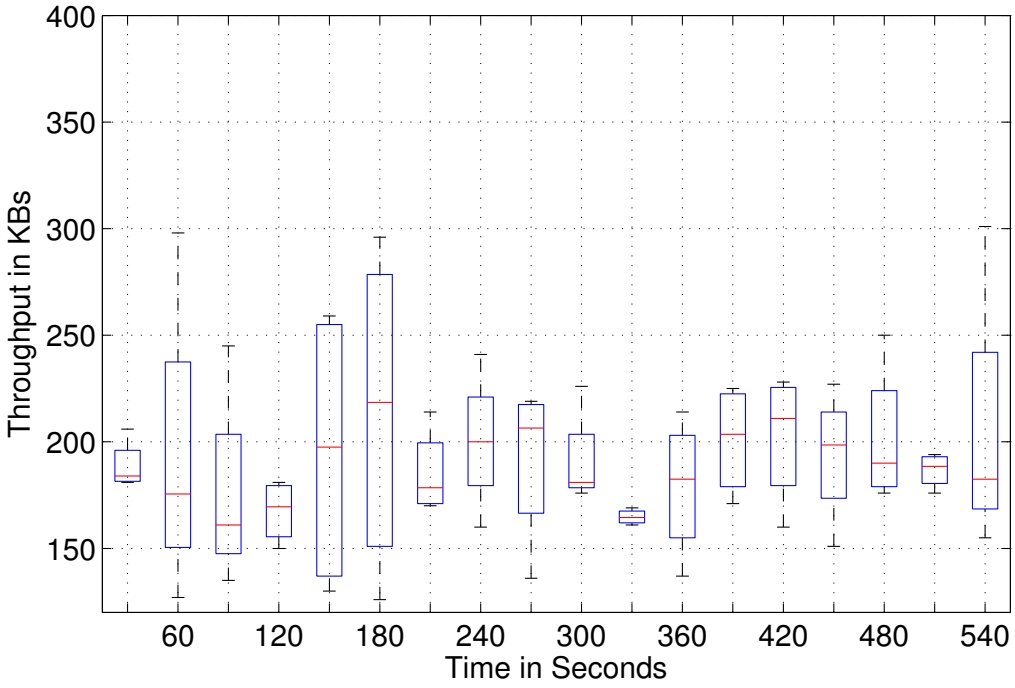


Figure 2.16: Single Stream Cubic TCP Throughput

2.5 Conclusions and Future Directions for RDRA

RDRA is a parallel TCP system with a fairness mechanism built into it in order to gain the robustness of parallel TCP without suffering from the unfairness caused by parallel TCP streams. RDRA is completely receiver based preserving the ease of use of parallel TCP by not requiring any in-network or sender side changes. I have demonstrated that RDRA develops about twice as much throughput as single stream TCP against packet loss while remaining fair with a single stream TCP. RDRA achieves all of the advantages of parallel TCP (throughput) while eliminating the disadvantages (unfairness). I have examined RDRA's queuing characteristics and found them to be roughly equivalent to a single stream TCP (with the exception of some additional jitter). Finally I tested RDRA and found that it performed well on real wireless

hardware.

RDRA is completely modular and can be upgraded and modified to fit individual needs. In particular RDRA has a pluggable congestion control mechanism and can be upgraded to use other forms of loss based TCP. In future work (that I have already conducted and describe in Chapters 4 and 5) I have investigated methods of determining the true RTT of an Internet path thus allowing RDRA congestion control mechanisms that relay on RTT such as delay based TCP variants. In addition using the RTT estimate as well as techniques described in Chapter 6 RDRA could be used to control queuing latency from the receiver.

RDRA is a parallel TCP system and as such it is suceptable to the pitfalls found in other parallel TCP systems. The two drawbacks are added queuing varablility shown in Figure 2.13 and initial window (IW) sizing. Parallel TCP does indeed add to the queuing variability, however, the variablity found in the queues of modern equipment is quite large even without parallel TCP. Short of the entire Internet switching to a less variable system of transmitting data such as paced TCP this problem is intractable and we should just learn to live with the variablity, [89]. The IW sizing problem occurs because the number of packets injected into the window on connection startup is the number of streams times the IW size. This can result in a large number of packets hitting the network queues all at once and lead to queue latency. The solution to the IW problem (for future work) is to add a slow start phase to RDRA's algorithm and to set the IW size to $\frac{IW}{n}$.

In addition, it should be noted that research efforts along similar tracks have since come to our attention. The earliest of these is SCTP a transport level protocol that uses parallel streaming, [65]. SCTP has not seen wide adoption because of the difficulty in routing the streams. Since our RDRA work was published there have been two large research efforts using parallel streaming from the client side at the application layer.

These are called SPDY and HTTP2.[6] SPDY had strong buy-in from Google and was proposed as a standard, however, it was replaced with its successor HTTP2 (also in RFC draft) before being adopted by the IETF. HTTP2 is very likely to become a published RFC and has strong adoption from both Microsoft and Google.

[6]https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2/

Chapter 3

The Fast Wireless Protocol (FWP) Algorithm

The robust and efficient streaming of video over wireless networks poses serious challenges. Inherent instabilities in the wireless medium lead to large, highly variable delay, throughput variations, and data loss. To cope with these problems, each layer of the network stack provides its own varying forms of protection strategy. However, this layered strategy often does not provide the best overall strategy. A protection mechanism at one layer may limit the operation of mechanisms at other layers. In this chapter, I identify and perform analysis on a major source of lost throughput in 802.11nac transmission aggregation systems. As a solution to this problem, I propose a novel cross layer network stack design using a dirty slate approach. Our dirty slate approach requires no server side changes or encodings whatsoever. It is constructed completely within the receiver's administrative domain. I demonstrate the throughput advantages of our approach in testbed and emulation and analyze the effects of overhead created by our system.

There has been much work in attempting to deliver video over wireless, and in particular, video over HTTP/TCP over wireless. What makes this an important, relevant, and (again) timely topic is the evolution in wireless technology and the demand

for video at high resolution. Research solutions addressing the efficient utilization of throughput in wireless networks fall into a few general categories. The most popular of these being the new generation of DASH video streaming applications. DASH solutions adapt the quality of the video stream delivered in order to match the bit rate required for media playout to the available throughput provided by TCP [77]. Examples of this type of solution are Adobe HTTP Dynamic Streaming server[1], Adobe OSMF[2], Microsoft's IIS Smooth Streaming player,[3] and the proprietary protocols used by Netflix, Move Networks[4], and others.

Further work in this area includes an evaluation of Akamai HD Network for Dynamic Streaming of Flash over HTTP is provided by Cicco et al. [15]. Of particular interest in this study are the experiments showing how the player reacts to sharing the bottleneck router with a greedy TCP flow. In addition, a study by Akhshabi et al. compare the behavior of Microsoft Smooth Streaming, Adobe OSMF, and the Netflix player [1]. These solutions prevent video playout from stopping due to a lack of throughput. However, they do this by reducing the quality of the video stream, not by increasing the efficiency of the network stack.

Before the adoption of HTTP/TCP by video streaming service providers, much research was done to investigate alternatives to the TCP protocol. Prime examples of this type of solution are the Datagram Congestion Control Protocol (DCCP) [49, 50], and RTP/UDP [28, 74]. A cross-layer example of a UDP based protocol is provided in Krishnamachari et al. [59]. Although these types of solutions provide demonstrable results, they were not adopted for general use. It is a common practice for Internet service providers to use firewalling to drop UDP packets making non-HTTP solutions

[1]http://www.adobe.com/products/
[2]http://www.osmf.org/
[3]http://www.iis.net/media/experiencesmoothstreaming
[4]http://www.movenetworks.com/

unacceptable. However, some streaming applications such as Skype[5] will attempt to find an RTP/UDP connection before falling back to an HTTP/TCP stream.

Many adaptations to TCP's congestion control algorithms have been studied in [17, 29, 80, 42, 65, 61, 65]. However, these solutions require changes to the TCP sender as well as the receiver. It has proven difficult to convince large content providers to change their network stacks. Our solution works within the user kernel and can be deployed without such large scale changes. Equation based TCP friendly solutions were studied in, [11, 33]. However, these equations are only fair within a factor of 2. This is inadequate for TCP fairness. TCP fairness and benchmarks have been studied in [81, 82, 83, 84, 14, 73, 70]. Multipath TCP has been studied in, [92, 34]. These types of solutions attempt to increase throughput by using multiple paths through the Internet. Split TCP solutions separate channel condition loss from congestion loss thus increasing TCP throughput in lossy networks [41]. However, these solutions do not address the overhead problems in the wireless network stack.

Erasure coding and it's use with TCP have been studied by Luby, and Mitzenmacher et al., [58, 79]. TCP/NC applies network coding in a shim layer between the TCP and IP layers. This approach is powerful, but it's positioning in the network stack requires a large overhead $(5n+7)$ where n is the number of packets involved in a linear combination). This overhead is caused because of the variable size of TCP packets. Our solution works at the Session layer where fixed size blocks can be used thus avoiding almost all of this overhead. In addition, TCP/NC requires sender side kernel changes making it as difficult to deploy as the above TCP adaptations. The ossification of the TCP protocol has been studied and it has been found that TCP modifications that relay on TCP options have difficulty passing through proxies such as those found in many wireless networks, [37, 32, 72, 4, 51].

[5] www.skype.com

These solutions although effective in their own right all solve fundamentally different problems than our FWP system. None of these solutions effectively address the problems encountered in 802.11 frame aggregation. I believe that the best approach to this problem is to keep the data flowing to the application. However, 802.11 wireless interrupts the flow of data because of its in order delivery requirement. 802.11 implements a sliding window system of retransmission. The problem with this system is that frames lost early in the sequence cause the window to stop and wait for the missing frames. The stop and wait action of the sliding window interrupts the flow of data causing lost transmission opportunities. Our approach to solving this problem is to remove the sliding window mechanism and allow data to be delivered out of order up through the network stack. In fact I think that it is best to handle data reliability to the application layer. Fundamentally what I am proposing is a system where the MAC layer does not worry about reliability or in order delivery. The MAC layers job is to just deliver the data as fast as possible in whatever order it arrives. The sliding window never stops and transmission opportunities are never lost.

The trade off with our approach is that reliability comes from the application and this places additional load on the network because some of the data has to be retransmitted over the network. I weighed the benefits of our solution in terms of increased throughput from taking advantage of all transmission opportunities against the drawback of increased overhead in the network in our evaluation. In this chapter our contributions are to analyze the source of lost transmission opportunities caused by the sliding window mechanism in 802.11 wireless. I design our solution the Fast Wireless Protocol (FWP), I build a prototype of our system and evaluate it using a testbed that I constructed. I restricted FWP to HTTP/TCP type connections because most video content is available over HTTP/TCP and these connections easily traverse firewalls and NATs. For the sake of deployability our FWP system is implemented on

the receiver stack only. There are no changes to any node in the network other than the receiver and no special data encodings at the server. Since FWP is implemented without crossing administrative domains it can be deployed without the need to secure the cooperation of other administrative entities.

3.1 Background

The 802.11 retransmission scheme causes a tremendous amount of overhead at higher bit rates. This is because the radio header, the Short Interframe Spacing (SIFS) wait times, and the ACK transmission take up a significant amount of radio time. In order to to reduce this overhead two frame aggregation schemes were standardized in 802.11n. The Aggregate Mac Service Data Unit (AMSDU) system, and the Aggregate Mac Protocol Data Unit (AMPDU) system. Frame aggregation systems combine mac data units (either service or protocol) into an aggregate with a single header. This significantly reduces overhead. The A-MSDU aggregation system combines Mac Service Data Units (MSDU) into 7935 byte aggregates with one MAC header and the payload protected by a single CRC.

The A-MSDU frame aggregation system is very efficient in relatively error free channels. However in error prone channels it suffers. The single CRC error protection does not allow the decoding of bits that were received correctly and the entire aggregate must be retransmitted. Figure 3.1 demonstrates the A-MSDU aggregation system. In time series A frame 2 is lost. There is no per packet checksumming so the entire aggregate is undecodable. Nothing is received, the entire transmission sequence is overhead. In time series B all the frames are received correctly and the overhead to data ratio is good. A-MSDU frame aggregation suffers greatly from error prone conditions. The A-MPDU system has an error correction protocol to cope with this.

In the A-MPDU system up to 64 frames are aggregated into a single A-MPDU

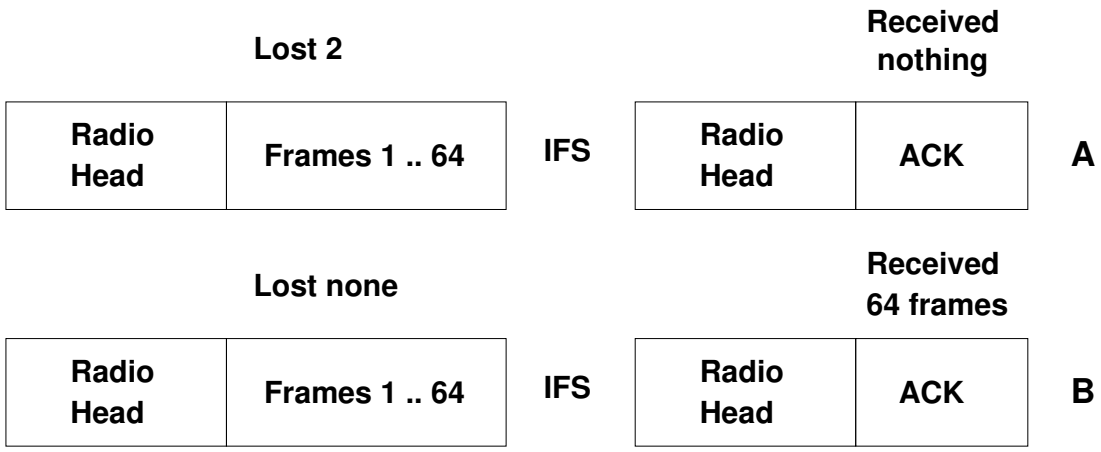


Figure 3.1: A-MSDU Frame Aggregation

with a single radio header. A checksum of 4 bytes is provided for each frame. With these checksum bytes the receiver can determine which packets were received correctly then generate a bit map called a Block Ack (BA) requesting the missing frames. The Block ACK Window (BAW) sliding window system is implemented to retransmit the requested missing frames.

Figure 3.2 demonstrates the BAW advance mechanism. In time series A the aggregate is full and 64 frames are transmitted. Frames 2 and 3 were not received correctly and the BA indicates that they should be retransmitted. The BAW sliding window cannot be advanced past the first missing frame until a BA indicates that frame has been successfully received. The BAW is advanced one frame. In time series B a smaller aggregate is transmitted. Only one new frame (number 65) can be added to the A-MPDU because the BAW was only advanced by one. The two lost frames can also be added for a total of 3 frames. The aggregate is now very small compared to the 64 frames that could have been transmitted. In this example frame 3 is lost again leading to another small aggregate transmission in time series C.

The A-MPDU system is very efficient even in error prone conditions. However, its sliding window retransmission system can sometimes stall resulting in the trans-

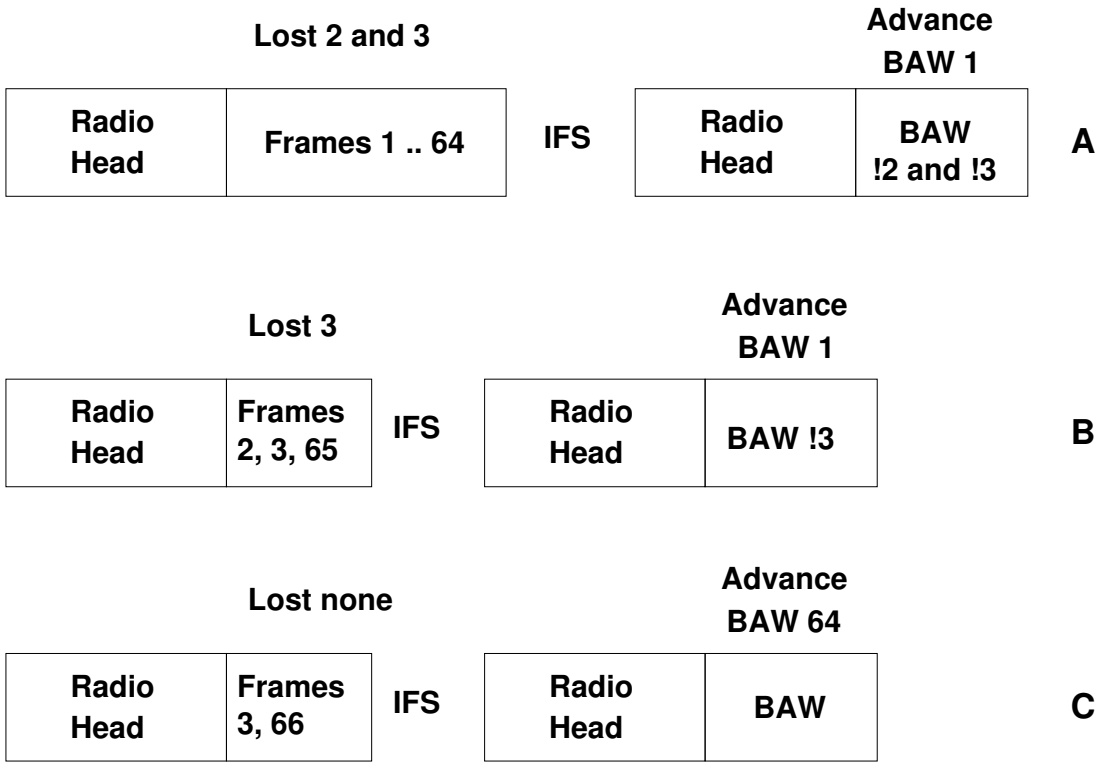


Figure 3.2: A-MPDU Block ACK Window Advance

mission of a small aggregate. This becomes a problem when there are other stations contending for air time. The station has used its transmission opportunity to send the small aggregate consisting of only a few frames. Now it must wait until it wins the contention for air time to transmit again. 802.11 wireless frame aggregation systems are very efficient compared to the transmission of a single frame at a time in the earlier systems. However, they have their drawbacks. A-MSDU aggregation is more effiecent than A-MPDU, but, it is only effective when the channel is nearly error free. A-MPDU aggregation is nearly as effiecient as A-MSDU, but, it transmits small A-MPDUs causing lost transmission opportunities and lost throughput.

3.2 Fast Wireless Protocol

Our approach to resolving the problems with 802.11 frame aggregation is to push the in order delivery and reliability services out of the MAC layer. This resolves the problem with the sliding window since there is no sliding window. The A-MPDU aggregate will always be filled and a transmission opportunity will never be lost. However, data will be delivered up the stack with missing frames.

The data will reach the transport layer with missing segments. This will be a problem for normal TCP connections. To resolve this I designed a receiver side modification to TCP called compatible TCP. Compatible TCP pushes the in order delivery and data reliability requirements up the stack to the session layer. It does this without upseting the congestion control system. The data will arrive at the session layer with chunks missing where the lost frames were. In order to cope with this problem I implemented an HTTP retransmission scheme that will replace the missing chunks with data. Once the retransmission is complete it will release the buffer and deliver the data to the application. The application remains unaware that the stack has been rearranged beneath it.

3.2.1 Requirements

In order to make the needs of our FWP system more concrete I have developed a set of requirements that I believe a solution must have.

1. Remove data protection and in order delivery services from the MAC/DLL.
2. Remove data protection and in order delivery services from the transport layer.
3. TCP congestion control services must remain intact.
4. Implement data protection and in order delivery services at the session layer.

Requirements 1 and 2 ensure that the flow of data is not inhibited by the lower layers. Requirement 3 ensures that the design is interoperable with other flavors of TCP. Requirement 4 ensures that the application is unaware of the new network stack. Requirement 1 is the most important. This ensures that no small aggregates will be sent and every transmission opportunity will be fully taken advantage of. I used A-MPDU aggregation with no ACKs. The receiver can request this behavior during the connection procedure by specifying ADDBA (add Block ACK) *noack* mode. ADDBA switches on the A-MPDUs, and noack turns off the acknowledgement system. Also I must reduce the MAC reorder buffer timeout to zero. These steps will satisfy requirement 1.

Our compatible TCP fulfills requirements 2 and 3. Removing the data protection and in order delivery services from TCP prevent the transport layer from confusing missing segments caused by lost frames with congestion and reducing the flow of data by mistake. TCP uses the same signal for congestion control as it does for data reliability. Because of this requirement 2 may interfere with the TCP congestion control mechanism. Requirement 3 ensures that congestion control remains intact allowing our compatible TCP to inter operate with other TCPs fairly. Requirement 4 implements our data protection and in order delivery systems. This requirements ensures that our rearrangement of the stack remains transparent to the HTTP over TCP video streaming application. This design decision produces overhead since it retransmits the lost data over the entire path rather than just the wireless hop.

3.3 FWP Implementation

I implemented a prototype in order to facilitate our evaluation and to make practical the conceptual framework described in Section 3.2.

Our FWP prototype design consists of four components.

1. An 802.11 FWP aggregation emulator. Described in subsection 3.3.2
2. An 802.11 A-MPDU aggregation emulator. Described in subsection 3.3.1.
3. A compatible TCP. Described in subsection 3.3.3.
4. An HTTP retransmission scheme. Described in subsection 3.3.4.

The 802.11 A-MPDU aggregation emulator works with the unmodified network stack. The 802.11 FWP aggregation emulator, however, delivers data to the transport layer with missing segments. Our compatible TCP is required to resolve this problem. Our compatible TCP delivers data with missing chunks. Our HTTP retransmission scheme resolves this problem.

I decided to use an emulator rather than 802.11n drivers and hardware for our experiments. I made this choice because emulation allowed us to study the effects of 802.11 frame aggregation without interference from other 802.11 systems. This is difficult to accomplish with wireless hardware because wireless standards have multiple systems interacting with each other. The rate adaptation system interacts with the driver to construct the transmit retry chain, and both of these systems interact with the packet aggregation system. These interactions between systems are driven by external effects that are difficult to control in an experiment. In addition, the emulators let us compare the systems while they are in a constant state. This is critical because I need to be certain that any effects observed in our experiments are not caused by seemingly random external effects. For instance if there is a throughput change I need to be certain that it was not caused by a Modulation and Coding Scheme (MCS) change or other system interaction.

I implemented our emulator as a packet scheduler running as a kernel module at the IP layer. I implemented 4 MIMO streams with channel error calculated independently on each stream. The channel error rate on each stream is the same in order to facilitate channel contention experiments. I set MTU size to 1500 bytes making a packet about equal to a frame. Packets are enqueued to each aggregation queue (1 per MIMO stream) round robin. I implemented two modes of operation, FWP, and A-MPDU. I also built an A-MSDU aggregation emulator. However, the performance of A-MSDU aggregation was so poor that I will not highlight any of the experiments in order to save space. Also I found that in the Atheros 802.11n drivers that A-MSDU aggregation is not implemented and I suspect that is the case for other drivers as well.

In the dequeue function of the emulator I implemented the aggregation. When enough packets have been enqueued to an aggregation queue to fill an aggregate (or to fill it as much as possible in the case of an A-MPDU) the aggregate is delayed to account for the overhead of the radio header, IFS, and ACK time. Frame loss is calculated, then successfull packets are sent across 1 Gbps Ethernet. Many simulators use a Bit Error Rate (BER) curve to calculate frame loss over wireless channels. However, since aggregations do not contain error correction bits (only per frame checksums) this is unnecessary. To streamline in kernel calculation, I used the Frame Error Rate (FER) model instead. Our designs are based on the open source Atheros driver code and the IEEE 802.11n standard. Other drivers are proprietary, however, I believe that they behave in a similar fashion.

3.3.1 802.11 FWP Aggregation Emulator

Requirement 1, Section 3.2 calls for the wireless driver to operate in A-MPDU mode with no ACKs, this is called ADDBA noack mode. I emulated this by queuing

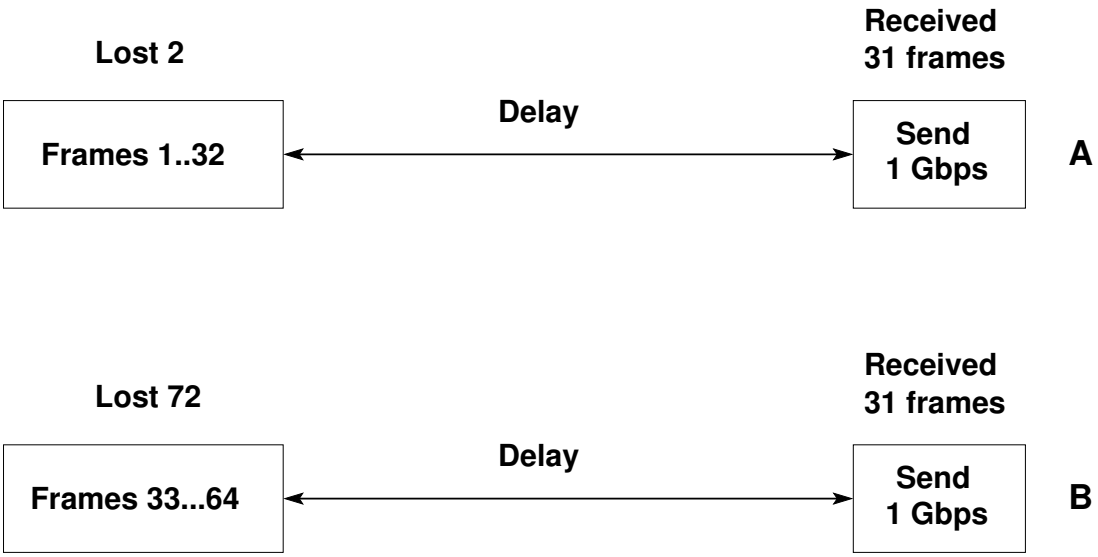


Figure 3.3: FWP 802.11 emulator using ADDBA noack

packets into the aggregation queues. The maximum A-MPDU aggregation size in the Atheros driver code is 32 rather than the 64 specified in the IEEE standard. I used 32 packets to an aggregate following the Atheros code. Frame loss is calculated using the FER. Packets representing lost frames are dropped. Packets representing successful frames are transmitted across 1 Gbps Ethernet after an appropriate delay for wireless transmission. This is demonstrated in Figure 3.3.

In time series A enough packets to represent 32 frames are queued. Frame 2 is lost so it's packet is dropped. A total of 31 frames are received. In time series B the aggregate is once again filled with 32 packets and the process is repeated with 31 more frames received. The aggregate is always filled and there is no sliding window. The throughput with FWP is always linear with frame loss because frame errors are not corrected.

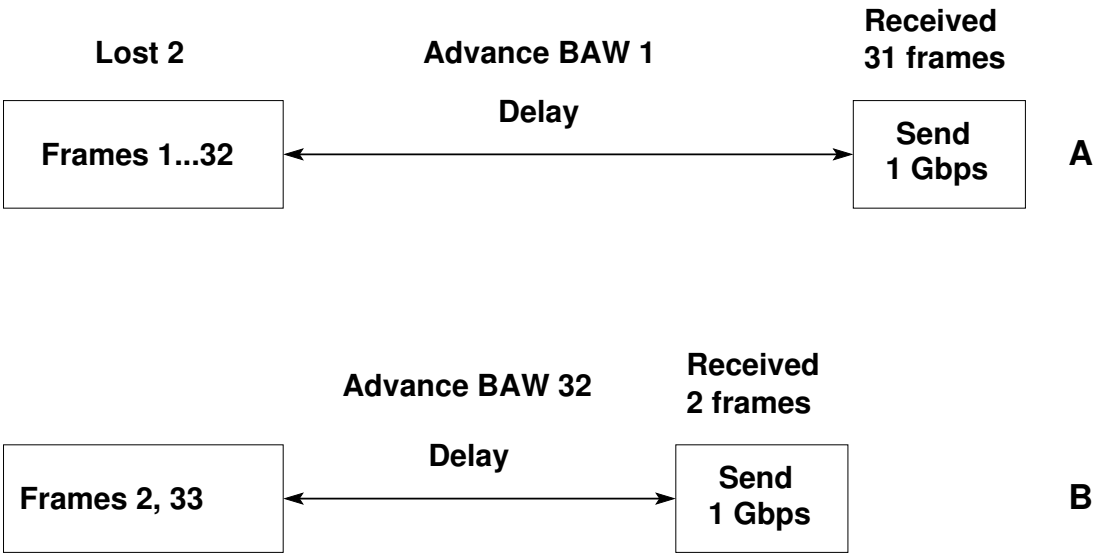


Figure 3.4: A-MPDU 802.11 emulator

3.3.2 802.11 A-MPDU Aggregation Emulator

In our A-MPDU emulator frame loss is also calculated using FER. However this time, packets representing lost frames are not dropped. Instead they are held in the queue. Packets representing successfully transmitted frames are delayed for overhead then transmitted over 1 Gbps Ethernet. In the next aggregate the BAW is advanced up to the first lost frame. New packets are added to the aggregate queue to account for the BAW advance. The aggregation queue now holds these new packets as well as any packets representing lost frames from the last round.

Figure 3.4 demonstrates this. In time series A frame 2 is lost and the BAW is advanced 1. The packet representing frame 2 is not transmitted. The successful frames are delayed and transmitted over 1 Gbps Ethernet. In time series B frame 2 is still in the queue from last time. The BAW has advanced one so the end of the window now points to 33. Packets representing frames 2 and 33 are transmitted. Both frames are successful so the packets are delayed for overhead and then dequeued.

The A-MPDU in time series B represents a lost transmission opportunity. If there

is no contention then this will not matter much and the A-MPDU system will achieve high performance. However, in the normal case when there is contention for the wireless medium the station will not be able to regain airtime until it wins contention again. In contention with other stations the A-MPDU system will not achieve the same high performance.

3.3.3 Compatible TCP

A standard TCP such as Cubic, Compound, or New Reno will not work well with our FWP system because it delivers data to the transport layer with missing segments. In fact, these TCPs are quite sensitive to sequence number holes. When a sequence number hole occurs dupACKs are sent for the missing segment until it is received. It takes approximately one Round Trip Time (RTT) to retrieve a missing segment. In normal use with RTTs above 30 ms many dupACKs will be sent before the missing segment is received. This behavior erroneously triggers the congestion control mechanism when segments are lost due to wireless transmission through the 802.11 FWP Aggregation Emulator.

In order to cope with this problem I developed compatible TCP. In keeping with our dirty slate design philosophy the server side transport layer code remains untouched implementing whatever congestion control mechanism is selected in the server kernel. Compatible TCP will fulfill requirements 2, and 3 from Section 3.2. In order to fulfill requirement 2 (remove in order delivery, and reliability) I monitored the TCP ACK stream. When a duplicate ACK is detected a placeholder segment is generated. The sequence number of the placeholder segment is fixed to the sequence number of the missing segment, and the DATA section is filled with marker data to indicate that the data in this segment was not received. The placeholder segment is then checksummed

and injected into the stack filling in the sequence number hole. This prevents TCP from detecting missing segments removing data protection and in order delivery services from the transport layer as specified in requirement 2 from Section 3.2.

Requirement 3 specifies that TCP congestion control must remain intact. TCP congestion control not only prevents congestion collapse but also maintains fairness with other TCPs. In order to re-establish the congestion control mechanism I operated our compatible TCP in a bi-stable mode. In one mode the compatible TCP injects placeholder packets, and in the other state normal TCP behavior is observed. The 802.11n rate adaptation system provides us with the expected frame loss for our current Modulation and Coding Scheme (MCS). If the loss rate measured over a window of 1 RTT is less than the expected frame loss rate then compatible TCP operates in placeholder injection mode. If the loss rate exceeds the expected frame loss rate then compatible TCP operates in normal TCP mode. An example of this is shown in Figure 3.5. Segment 3 is lost and filled in with a placeholder as soon as the sequence number hole is detected (when segment 4 arrives). At segment 9 the sequence number hole count has exceeded the frame error rate reported by the 802.11n rate adaptation system. In response compatible TCP switches to normal TCP mode. No placeholders are injected and duplicate ACKs are sent triggering a congestion event.

3.3.4 HTTP retransmission scheme

The final requirement from Section 3.2 implements in order delivery and data protection services. This allows applications to remain unaware of the rearrangement in the stack below them. I used an HTTP retransmission scheme in order to accomplish this. The placeholder packets are easily recognized in the data stream by a character search. The beginning and the end of the placeholder data indicate the byte range

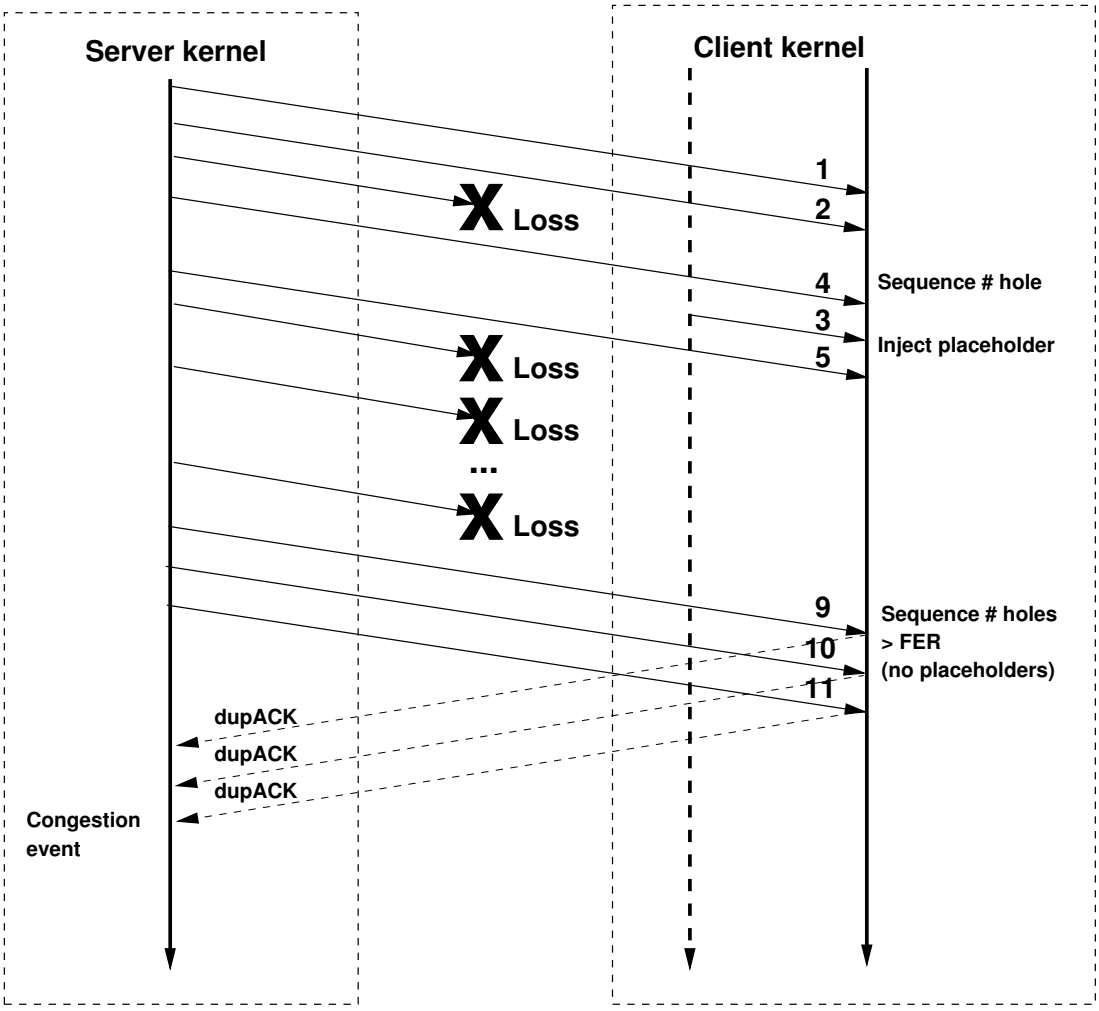


Figure 3.5: TCP Compatible Transport

that must be requested to replace the missing data. I used curl[6] to request the byte range and insert the missing data into the stream before releasing the buffer to the application.

[6]http://curl.haxx.se/

3.4 Evaluation

I evaluated FWP by observing three key characteristics of our solution, throughput, TCP interoperability (fairness), and overhead. FWP is designed to take full advantage of every transmit opportunity and achieve better throughput than 802.11 frame aggregation during contention. I evaluated the throughput gains of our solution over varying channel conditions and number of competing stations.

In our design I replaced the receiving side TCP with our compatible TCP. To determine interoperability with other TCPs I performed fairness testing. I introduced a variant of Jains fairness metric to determine whether our TCP solution shares fairly with other TCPs. In addition I characterized the overhead introduced by our HTTP retransmission scheme. Although the network stacks are real our wireless hardware is emulated. I understand that emulators have difficulty modeling the complex interactions of the wireless channel. Because of this I do not rely on our emulator to provide absolute values, but instead use it to understand behaviors and trends that cannot be observed in isolation with real hardware.

3.4.1 Testbed

In order to evaluate our FWP solution and compare it against A-MPDU frame aggregation I built a testbed in Emulab facilities provided by the Flux Group, part of the School of Computing at the University of Utah[7]. Nodes 1 through m in Figure 3.6 are client nodes. Node m is equipped with an FWP stack. Because Emulab topologies are easily configurable I could vary the number of clients to fit the needs of the experiments. This allows me to test contention with other nodes in our testbed. The servers (nodes m+1 through n) are a mirror image of the clients with 1 server per

[7]https://www.emulab.net/

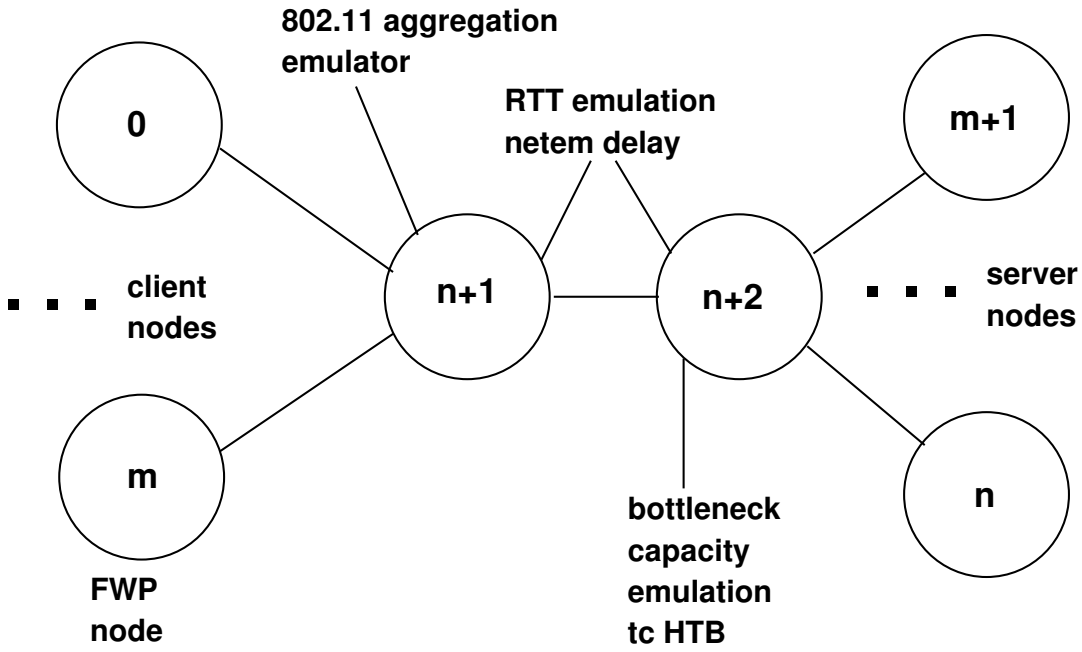


Figure 3.6: Emulab testbed

client. Nodes n+1 and n+2 emulate the Internet path from each server to each client. All links are 1 Gbps Ethernet.

Because the maximum throughput of a path through the Internet is determined by the capacity at the bottleneck router I implemented rate control. I used tc[8] filters and an HTB packet scheduler[9]. RTT is implemented using netem[10] to introduce one way delay on the egress interface of nodes n+1 and n+2.

Table 3.1 shows a list of default experimental parameters. Unless otherwise specified in an experiment the parameter values will be set as shown in the table. I chose a default RTT of 40 ms. Although the average RTT of an Internet path is a nebulous and debatable point our observations have shown that RTTs between 30 and 50 ms are within reason. Because of the interaction of multiple router queues along an Internet path RTTs in the Internet are not stable values. Since netem is capable of randomly

[8]http://www.lartc.org/
[9]http://linux.die.net/man/8/tc-htb
[10]http://www.linuxfoundation.org/networking/netem

Experiment Parameter	Default Value
Round Trip Time	40 ms
RTT Variance	10 percent
Bottleneck Throughput	600 Mbps
Frame Loss	$10^{-3} - 10^{-2}$
Experiment Duration	120 seconds
Experiment Runs	10

Table 3.1: Default Experimental Parameters

varying the delay according to a distribution I chose to vary the RTTs +/-10% in a normal distribution about the mean. I chose to use 600 Mbps throughput capacity at the bottleneck since this is the theoretical maximum of 802.11n. Our links are 1 Gbps so it was not possible for us to measure more than one 802.11n station operating at full theoretical maximum. Experiments were run for 2 minutes with 10 experimental runs.

3.4.2 Throughput

To understand the throughput gains achieved by our FWP system I first highlight an example from a series of experiments with 10 competing stations sharing 600 Mbps of throughput. Figure 3.7 shows a comparison of our FWP versus 802.11 frame aggregation measuring throughput achieved at the receiving station against probability of frame loss. As expected the throughput achieved by our FWP system is very linear to loss. This is because I transmited a full (32 frame) aggregate every time FWP won contention and do not stop to retransmit. The A-MPDU system on the other hand is not linear to loss. The A-MPDU aggregation system suffers a drastic reduction of throughput at even very small error rates. It achieves about one third of the throughput of FWP at .05 FER, and about half at .20 FER probability.

The emulation shows that the trending throughput gains are significant especially at

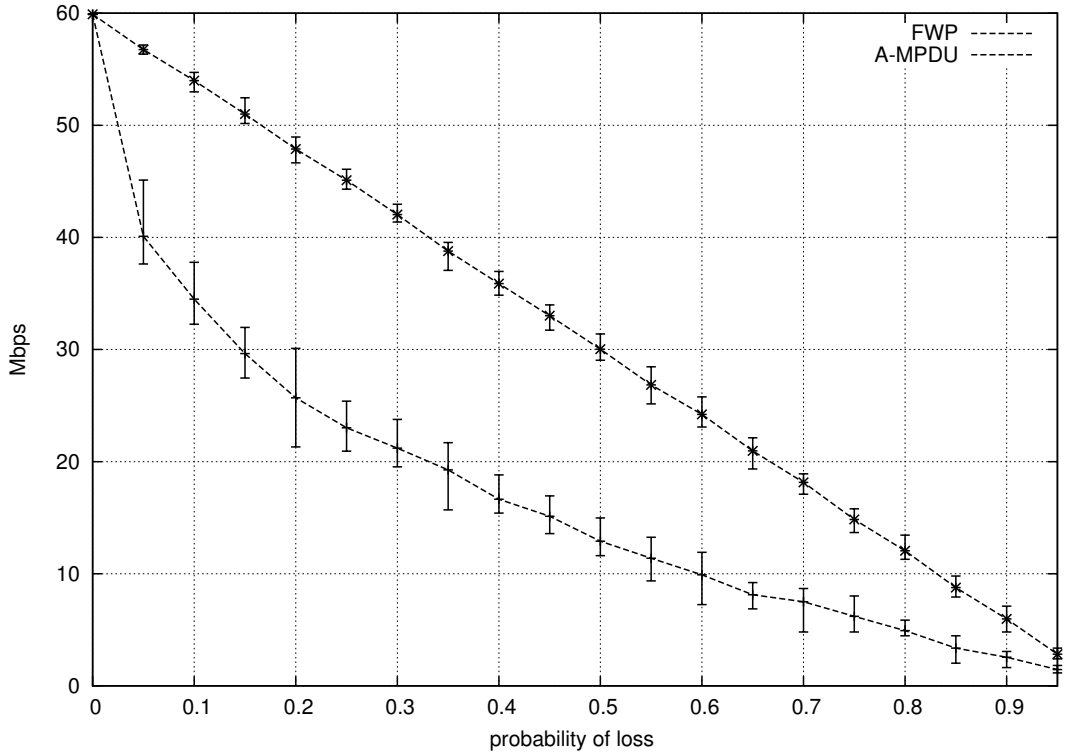


Figure 3.7: Throughput gains from 1 FWP station competing with 9 A-MPDU aggregation stations

the FER range of 10% to 40%. This range of FER is critical because this is where a rate adaptation system might make a decision such as 40% FER at 600 Mbps is preferable to 0% FER at 300 Mbps. Next I sought to determine the effect of the number of competing stations on our FWP system versus A-MPDU aggregation. Since I know from our experiments that with no competing stations the throughput achieved by each system is the same I plotted the throughput gains achieved by 1 FWP station competing with 1 to 9 A-MPDU stations.

I defined a metric called speedup to measure the throughput gains. Speedup is the number of times faster that FWP is than A-MPDU aggregation. For instance a speedup of 1 would be the same throughput, a speedup of 2 would indicate twice as much throughput. The experiments were each two minutes in duration. The graph

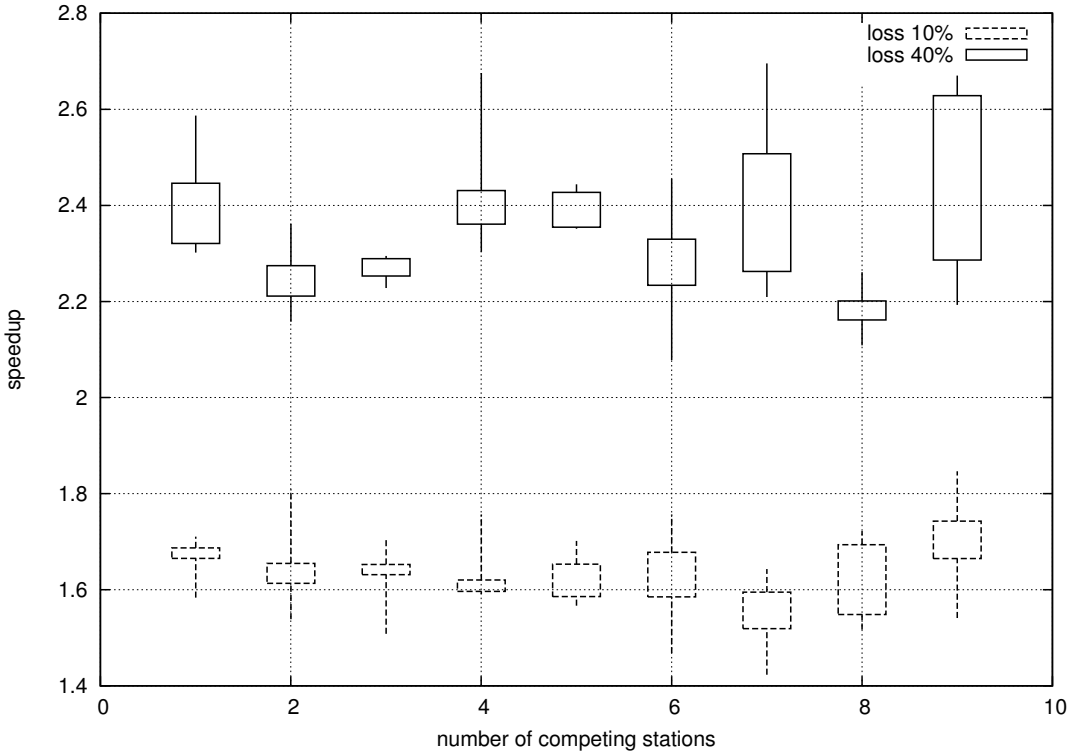


Figure 3.8: Speedup of FWP against number of competing A-MPDU stations

in Figure 3.8 shows a whisker plot of 10 experimental runs. The bars in the whisker plot show values from the 10th to the 75th percentile, and the whiskers show the minimum and maximum values. Figure 3.8 shows that the throughput gains are more strongly effected by FER (loss) than by the number of competing stations. In fact the throughput gains are reasonably flat across the number of competing stations. They are averaging greater than 1.6 for 10% FER, and greater than 2 for 40% FER.

These experiments that I have highlighted in this chapter demonstrate that the throughput gains of FWP over current wireless aggregation technology are significant and not dependent on the number of competing stations. This indicates that in real wireless hardware FWP would almost always (any non zero channel error condition) develop significant throughput gains and that these gains would increase with the amount of channel error.

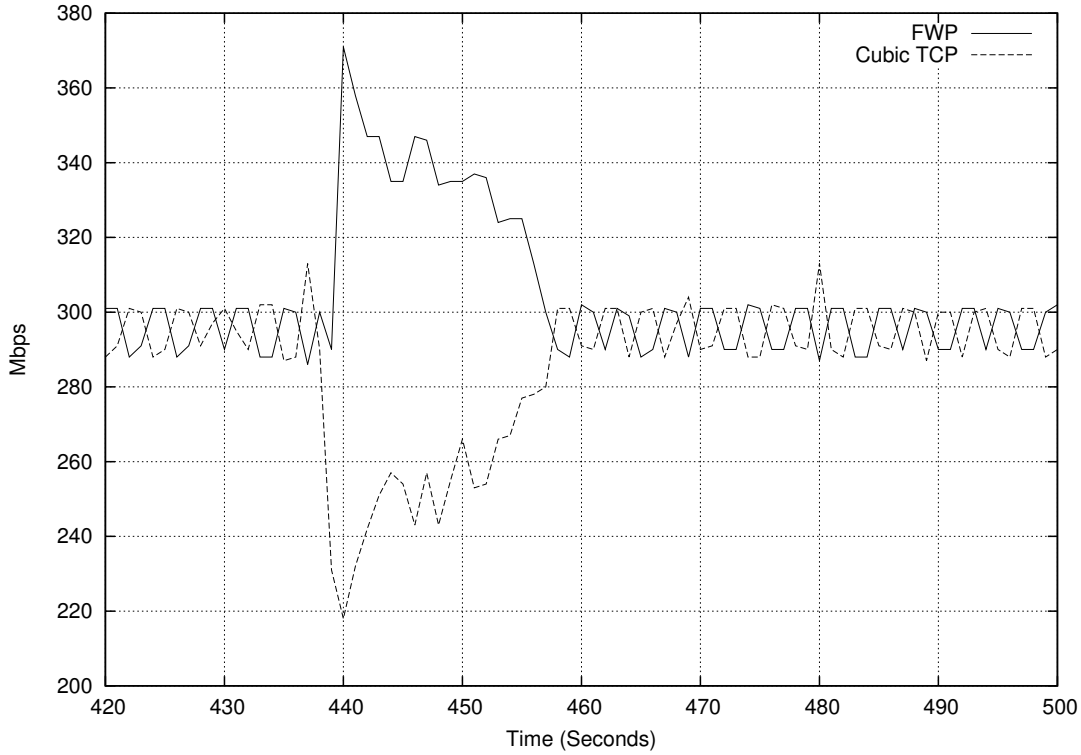


Figure 3.9: Compatible TCP competing with Cubic over a time series

3.4.3 Fairness

On of the key goals of our dirty slate design for FWP is deployability. Because of this I had to determine whether our compatible TCP operates fairly with other TCPs. First I wished to determine if the characteristic competitive behavior of TCP has been affected by our modifications. This involved many time series experiments to determine that the waveform generated by our compatible TCP is different from the waveform generated by a popular TCP such as Cubic.

In Figure 3.9 shows an excerpt from one of this series of experiments. The bottleneck router capacity in this experiment was 600 Mbps and the RTT was 40 ms. The graph shows that the competitive behavior of TCP remains intact in our compatible TCP. One flow gains an advantage over the other for a time and then the roles reverse.

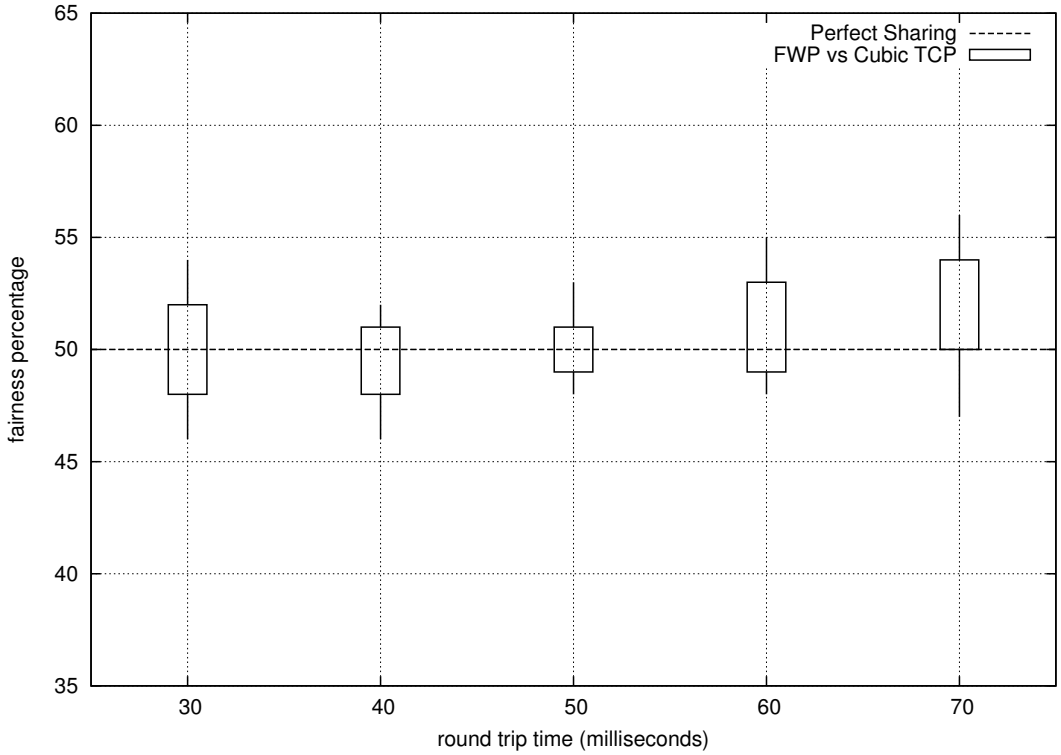


Figure 3.10: Throughput variations of compatible TCP against TCP Cubic

Normally the variations in throughput are quite small but sometimes over larger periods of time the fluctuations are larger. These experiments show that the competitive behavior of TCP is largely unaffected by our modification. In the next series of experiments I performed a more rigorous system testing of TCP fairness over a variety of RTTs. In order to clearly present these experiments I defined a metric called fairness. This is loosely based on Jains fairness index.

I ran this series of experiment for 2 minute durations and determined the variations in throughput over 1 second intervals. The bottleneck router capacity in these experiments was 600 Mbps and I plotted the percentage of throughput achieved by our compatible TCP over a competing Cubic TCP. The design of our compatible TCP can take up to an RTT longer than a normal TCP to begin congestion control behavior. This could lead to more aggressive behavior over larger RTTs. I believe that it is okay

for a TCP to act in a more aggressive manner when the RTT is larger because TCP throughput gets smaller as RTTs increase. However, the amount of aggression should not be too great at normal RTTs.

The whisker plot in Figure 3.10 shows our fairness percentage results over 10 experimental runs. The line at 50% fairness indicates "perfect sharing". This would mean that both TCPs (compatible and Cubic) received 1 half of the throughput over each 1 second interval. A "fair" TCP should not fluctuate much above or below this perfect sharing line. Plus or minus 5% would indicate very good sharing, and +/- 10 % would still be a very reasonable amount of fairness.

The graph shows that our compatible TCP operates in a reasonably fair manner when competing with a Cubic TCP across a wide range of RTTs from 30 to 70 ms. There is a small amount of additional aggression at the higher RTTs (60 - 70 ms), however, the sharing of bottleneck router resources is still very reasonable. These experiments have shown that our compatible TCP behaves in a manner consistent with standard TCP behavior and that it is reasonably fair with other TCPs (TCP Cubic). I believe that our compatible TCP is interoperable with other TCPs fulfilling requirement 3 from Section 3.2.

3.4.4 Overhead

One of the tradeoffs with our FWP solution is that the HTTP retransmission reliability system generates overhead. There are two types of overhead that I examined in our experimentation. The first is the additional data transfer across the wireless link. The overhead across the wireless portion of the link is small because the data portion would have had to have been retransmitted anyways. The only additional overhead is the HTTP header required to specify which portion of the data needs to

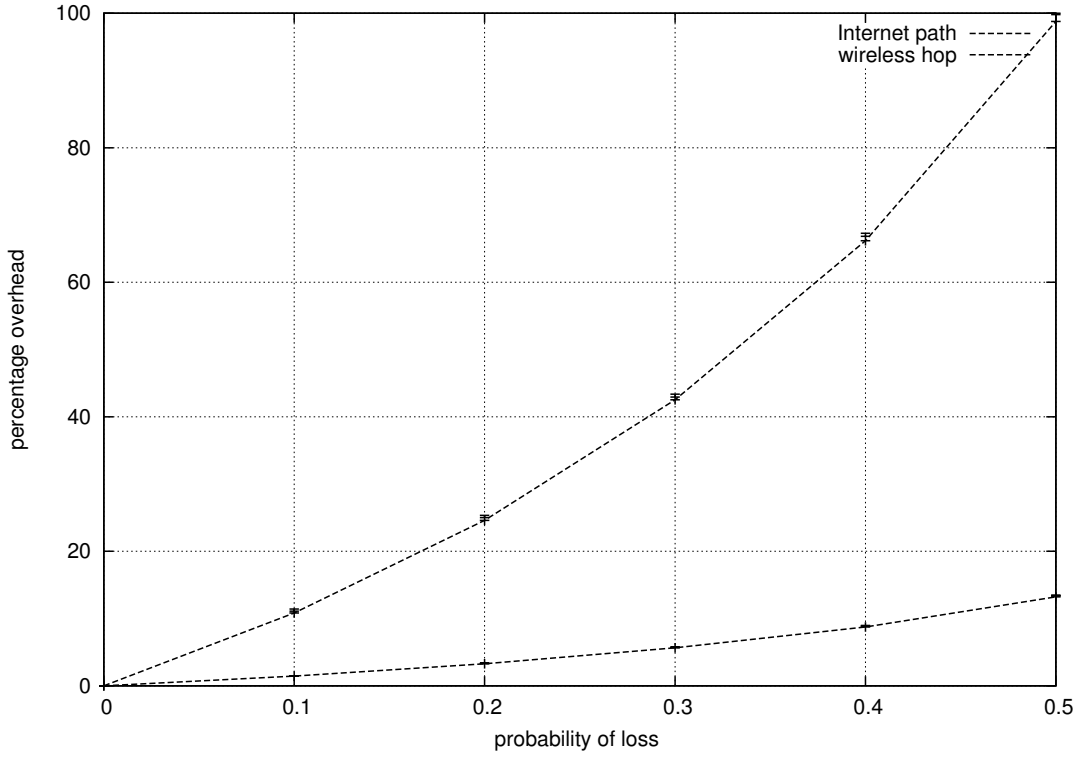


Figure 3.11: Overhead for Internet path versus wireless hop

be retransmitted.

The second type of overhead that I examined is the amount additional data transfer across the Internet path. Because our HTTP retransmission scheme retrieves missing data from the server this data must be sent all the way across the path. For every other hop on the path the retransmitted data is pure overhead since the data would have been transmitted from the access router instead of the server. Figure 3.11 shows the comparison of both kinds of overhead. The overhead across the Internet path grows quickly reaching 100% at a 50% FER. This shows that our FWP system places a large burden on the Internet path rather than the wireless hop.

The overhead for the wireless hop however is quite small. It only reaches 13% at 50% FER. These experiments have shown that our FWP system provides a great deal of throughput gain versus overhead from the point of view of the wireless hop.

However, the increasing overhead across the Internet path becomes a limiting factor. I plan to address this problem in future work.

3.5 Conclusions and Future Work

In this work I have developed FWP, a very fast wireless protocol. I have done this using our novel "dirty slate" architecture. The system does not require any in-network or server side changes whatsoever. In addition, it does not require any special encodings or data formats. Our "dirty slate" architecture solves the deployment problems encountered by previous work in this area.

However, I found that even though the amount of overhead introduced into the wireless hop is minimal, the amount of overhead introduced into the Internet path is large. The true contribution of this work is to separate the losses that occure because of channel related causes and congestion related causes. Because congestion related losses occur before ever reaching the wireless hop there is no frame number sequence hole caused by congestion related segment losses. Congestion related losses are preserved for TCP and channel related losses are "skipped over". In future work the overhead of frame retransmission could be addressed by allowing the 802.11 re-transmission scheme to retrieve the missing frame. These frames would then be grabbed and sent via $netfilterqueue$ to the session layer for possible reintegration into the flow.

This possible reintegration of missing data requires smart applications that can determine whether or not they still want the packet. This is desirable for deadline driven applications such as live VOIP and video which have no use for the replacement of missing packets after the playout deadline has passed. These types of applications would want the ability to release the buffer that is waiting for the missing data if they have determined that the data is no longer valuable to them. In addition, the concept

of “look-ahead” where an application can look at the recieved data while it is waiting for the retransmission to occur. This provides an application the ability to get started working on data that it might find useful before the retransmission has occured.

Chapter 4

A Cross-Disciplinary Approach to Queue Sizing

4.1 Introduction

The queue sizing problem spans the entire Internet, it has been with us since the beginning of packet switched networks and will be with us into the foreseeable future. In order to understand the queue sizing problem it is necessary to take a cross-disciplinary approach. This cross-disciplinary approach is necessary because no single discipline encompasses the breadth of knowledge needed to approach the problem in a holistic manner. The best way to accomplish is to gather together into one body of knowledge an amalgamation of disciplines in the networking sciences from the following disciplines; queuing theory, networking topology, network sensing, transport and queue management.

The body of knowledge encompassed by this chapter is a sufficient and necessary amalgamation of intimately interconnected knowledge from a variety of networking science disciplines. Queuing theory is necessary because it provides the essential equations that our algorithms are based on. Networking topology gives us an understanding of where and when to apply the algorithms. Network sensing and transport provide

the essential data that the algorithms require for input and queue management demonstrates the requisite methodology needed to control the queue size. Packet scheduling provides the class based queuing that is necessary in order to separate flows with unique and individual queue sizing needs into their own queues. Queue management provides the basic techniques that are needed to control queue size. I begin with queuing theory and network topology.

4.2 Queuing Theory and Network Topology

I grouped queuing theory and networking topology concepts together because queuing theory provides the indispensable equations needed to address the queue sizing problem and network topology provides an understanding of where the algorithms must be applied.

4.2.1 Queuing Theory

Queuing theory provides the indispensable equations which drive the algorithms. The Internet is a massively diverse place that defies creating a unifying set of equations that work in the general case. This problem is made more tractable by separating the problem into two parts; the core and the edge. I discuss the difference between these parts in detail in Section 4.2.2. For the purposes of this chapter it suffices to know that the core consists of "high speed" routers that receive much traffic and generally are servicing many long term flows at any given time in addition to bursts of traffic. Core routers on the Internet are governed by $O\left(\frac{Capacity}{\sqrt{number of flows}}\right)$,Appenzellar et al. [3].

In our work I am focused on edge routers because this is where the throughput and delay experienced by an end user is typically controlled. Core routers also control their latency and , but, not with an individual customer in mind. The edge of the

Internet is much closer to the consumer (end user) and the routers are much slower and often have little or no traffic (though they can be saturated) because they serve few or even one customer. Flows through edge routers are governed by the bandwidth delay product equation, Villamizar et al. [86].

The queue sizing equation is elegant and straightforward $Q = B * D$ where Q is the queue size, B is the throughput and D is the delay or Round Trip Time (RTT). This is because the sending computer needs to be able to inject enough packets into the network to keep the slowest router in the path busy for one RTT. At the end of the RTT the ACKs will begin arriving and the sending host will transmit more. The correlation between queue size, throughput and delay is demonstrated by the following equations:

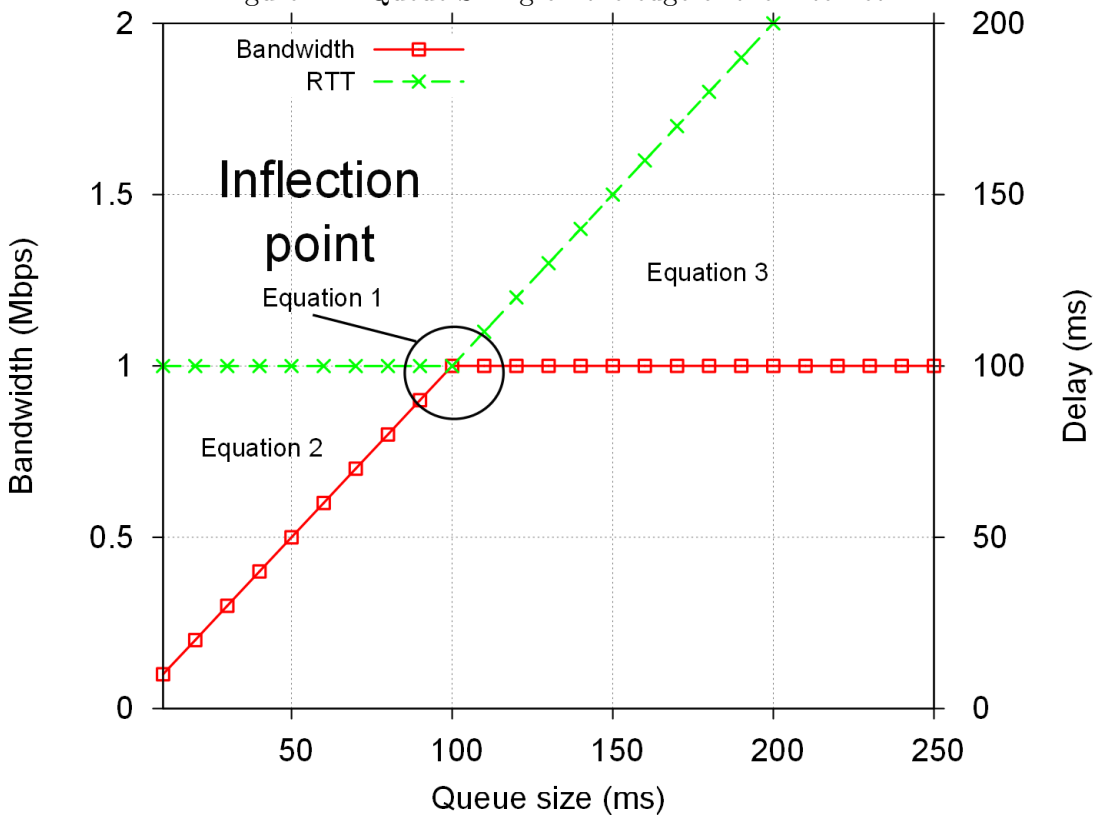
$$Q_{size} = B * D \rightarrow B = optimal; D = optimal \tag{4.1}$$

$$Q_{size} < B * D \rightarrow B < optimal; D = optimal \tag{4.2}$$

$$Q_{size} > B * D \rightarrow B = optimal; D > optimal \tag{4.3}$$

In the above equations Q_{size} is the queue size in bytes, B the throughput in bytes and D the delay in seconds. The left hand of the equations represent the condition (queue size is too small/large/correct) and the right hand side describes the effect upon the flow of data through this path. For a given flow when the delay is approximately equal to the Round Trip Time (RTT) times the throughput (the emission rate of the slowest router) the resulting throughput and delay will be optimal as described in Equation 4.1. If the queue size is less than the bandwidth delay product then the flow

Figure 4.1: Queue Sizing on the edge of the Internet



will lose throughput according to Equation 4.2. If the queue size is greater than the bandwidth delay product then the flow will experience excessive delay as described in Equation 4.3.

The queue sizing effects described in Equations 4.1, 4.2 and 4.3 are shown in the graph from Figure 4.1. The path characteristics described in the example are for a flow with 100 ms RTT at 1 Mbps in either the upload or the download direction (it does not matter for this example). The graph shows that when the queue size is 100 ms (12,500 bytes) the flow achieves its maximum throughput (1 Mbps) at the minimum delay (100 ms). This inflection point is circled in the graph from Figure 4.1 and the flow is operating according to Equation 4.1. When the queue size is smaller than 100 ms the flow loses throughput and the delay remains 100 ms. In this region of the graph the

flow is operating according to Equation 4.2. When the queue size is larger than 100 ms the flow achieves the full 1 Mbps throughput, but, delay increases above the 100 ms minimum. In this region of the graph the flow is operating according to Equation 4.3.

Clearly the correct queue size for a flow on the Internet is described by Equation 4.1. This applies to individual flows even though they flow through core routers governed by Applenzeller's equation [3]. The reason that Equation 4.1 applies is because queue buildup for an individual flow occurs at the slowest router in the path. Villamizar's equation applies to core routers because they have high throughput and service many flows. The throughput delay product equation applies to individual flows and occurs at the slowest router int the path (near the edge). The next step is to examine Internet topology in order to understand where these equations apply and why.

4.2.2 Network Topology

The topology and network neighborhood of a typical Internet connection is shown in Figure 4.2. In this dissertation I describe a connection that uses cable technology for the purposes of demonstration. However, the principles as applied to queue sizing are technology agnostic and apply to Digital Subscriber Line (DSL), cable, broadband, satellite and other packet switched technologies equally. Any Internet connection consists of a number of users using wired or wireless connections to connect to an access router. The access router connects to a modem owned by the Internet Service Provider (ISP) which connects to a Cable Modem Termination System (CMTS) or other device depending on the technology used by the ISP through an access link.

The access link has two properties that present unique opportunities for queue management. The access link is typically the slowest portion commonly called the primary bottleneck in any given Internet path because this is where the ISP rate

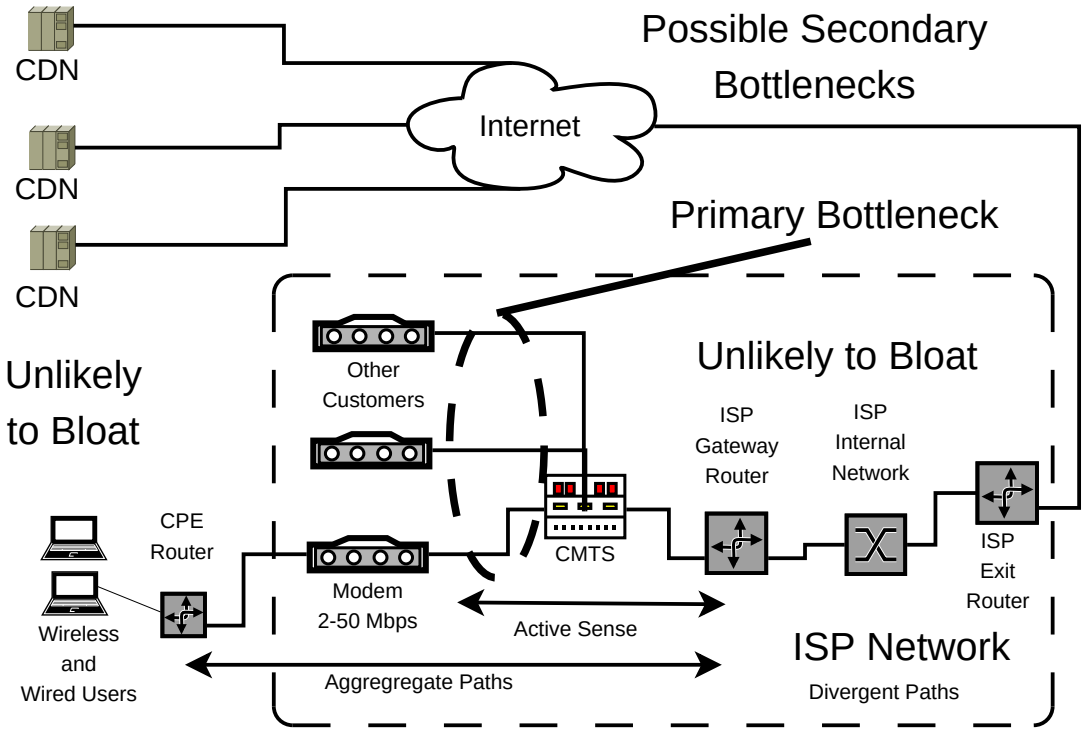


Figure 4.2: Typical Consumer Network Neighborhood

limiting is applied. Applying queue management at the slowest device in a path is important because this is where the Equations 4.1, 4.2 and 4.3 apply. The access link is also the place where all flows that share queues are present. This aggregation of flows property is important because it allows queue management to be applied to all flows equally. For instance, if the users from Figure 4.2 were to apply queue management individually then any user joining the network without queue management could fill the buffers at the modem and defeat the queue management of the other users.

Queue management is typically applied at the modem. I provide much more detail on the access link, the modem and the CMTS in Chapter 5, but, for this discussion I will simply refer to the modem as the slowest part of the path. The reason queue management is normally applied at the modem is because it is the slowest device where all flows sharing a queue are aggregated. The modem is the slowest point in

the path for all flows for all users on a given access link and the place where all flows are present. This set of properties presents a unique opportunity to apply queue management. The physical topology in figure 4.2 seems to indicate that flows from other modems aggregate at the CMTS. This is not the case since each access link is frequency separated from the others. Flows from separate modems do not share queues until they reach the ISP router. The queuing for a flow in an individual modem cannot be affected by a flow in another modem and all flows that can affect this flow are present where they can be managed according to policy.

There are two cases where the modem is not the slowest device in the path for any given flow. The first is when business disputes between providers result in congestion at interconnection points and network paths. These secondary bottlenecks shown in Figure 4.2 can cause performance degradation for users. This problem is out of scope for this chapter because there is nothing that an AQM algorithm can do about a secondary bottleneck caused by a business dispute. This problem is not scientific nor technical and it should be solved by solving the business dispute that is causing a content providers data path to be slowed down. The problem is in the domain of politics and business and likely will be resolved when the debate over net neutrality is resolved one way or another. I do however, in chapter 7 discuss methods to mitigate the queue sizing problems caused by the throughput reduction imposed.

The second case is where the access link is faster than the wireless or wired link to the access router. In most cases this type of problem can be fixed by upgrading from old technology. For instance if a user is running a wired connection over 1 Mbps Ethernet and the bottleneck moves into the 1 Mbps queues the answer is to simply upgrade to current technology. However, in cases where the access link is exceptionally fast (1 Gbps fiber) and wireless connectivity is desired. The Gigabit fiber may operate faster than any wireless technology available. These cases are atypical in the Internet

today and our solutions focus on the access link. However, in Chapter 7 I also discuss methods for adapting our technology to these cases.

4.3 Transport and Network Sensing

Transport and network sensing is necessary to solving the queue sizing problem because it is the interaction of the transport protocol with the network queue that causes the queue sizing problem. In addition, network sensing is necessary in order to gain information about the state of the network queue in order to manage the queue size to its proper size. I grouped these two disciplines together because they are so interrelated. Transport contains elements of network sensing and it controls the network queue size by increasing or decreasing the flow of data.

4.3.1 Transport

The Transport Control Protocol (TCP) is by far and away the most ubiquitous transport protocol deployed and in use today. Other protocols are in use such as the User Datagram Protocol (UDP) and I will address the handling of these protocols in Section 4.4. TCP transport comes in many flavors which can be divided into loss based and delay based TCP as well as hybrids between the two. Loss based TCPs control the flow of data by increasing the flow until a packet loss is detected. At that point the transport protocol decreases the flow of data draining the network queue of packets and begins to increase again.

Modern TCPs respond to Explicit Congestion Notification (ECN) signals as well as loss [32, 72]. ECN marks data packets before the queue is filled to the point of needing to drop packets. The ECN marked data packets flow to the TCP receiver and the TCP receiver reflects the ECN mark into the ACK. When the ECN marked ACK

is received by the sender the sending TCP responds to the ECN mark exactly as if a packet has been dropped. ECN marking allows the lossless transmission of data and is recommended by the Internet Engineering Task Force in RFC 3168.[1] Not all network devices support ECN marking and the fallback is packet dropping [4, 51].

Loss based TCP algorithms commonly deployed and active on the Internet today include Cubic (Linux), Compound (Windows XP), and NewReno (mac, Windows Vista+), [19, 68, 80]. Compound is a hybrid and responds to delay as well as loss. Other examples of loss based TCP algorithms include [17, 65, 61, 10, 56, 25].

Delay based TCP algorithms are sensitive to RTT and reduce queue size when delay starts to increase. Examples of delay based TCP algorithms are Vegas, CAIA CDG, and Fast, [36, 82, 9]. Delay based TCPs attempt to manage the queue size by themselves. However, the problem with this approach is that if a delay based TCP algorithm shares a queue with a loss based TCP algorithm then the delay based protocol will reduce its data flow in response to the increased delay caused by the loss based protocol. This causes the delay based to lose throughput to the loss based protocol. Unfortunately in the wild there is always a loss based protocol sharing the queue.

Other examples of TCP include equation based TCP, multipath TCP, network coding TCP, SCTP and split TCP [33, 34, 65, 41, 79]. Equation based TCP uses the loss and RTT to calculate the correct throughput, multipath is TCP using multiple paths, split TCP is a single connection split into two TCP sessions and network coding TCP uses coding in order to overcome error. In any case though these protocols react in different manners, they all attempt to fill the network queue to its maximum. As described in Section 4.2.1 each flow has a unique queue size according to its throughput and delay characteristics. The proper queue size must be set with respect to these

[1] https://tools.ietf.org/html/rfc3168

characteristics, not by simply filling the queue to its maximum size.

4.3.2 Network Sensing

There are many methods of sensing the network that measure various metrics in both active and passive modes. However, there are only two characteristics that are relevant to the queue sizing problem. These characteristics are the throughput and the RTT. I sense the network in two modes; actively and passively. Active measurement consists of injecting sense packets into the network and measuring them. Active measurement induces overhead caused by the injected sensory packets and creates a tradeoff between network overhead and sensory granularity. In general the finer the granularity and hence the more accurate the measurement the more overhead that is created. Passive sensing does not inject packets into the network and does not create additional network overhead. Instead passive sensory systems take measurements from data packets or ACKs that are already traveling the network path.

I measure throughput passively by measuring the length of outgoing data packets until a threshold level of transmitted bytes has been transmitted as well as the time required to transmit them. This value is then exponentially smoothed. This method is consistent with the throughput measurement for queue drain rate as described in the PIE algorithm [69]. I passively measure the RTT using TCP timestamps and the equations.[2]

$$RTT_{var} = (1 - beta) * RTT_{var} + beta * |SRTT - R| \tag{4.4}$$

$$SRTT = (1 - alpha) * RTT_{var} + alpha * R \tag{4.5}$$

[2]https://tools.ietf.org/html/rfc6298

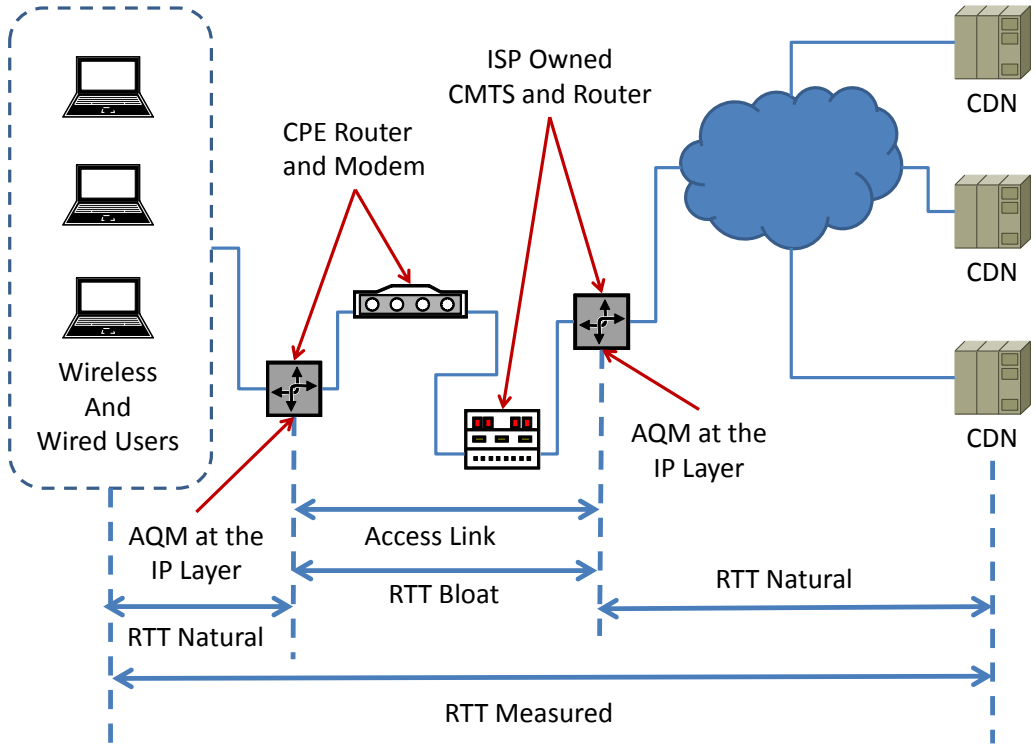
Where R is the RTT sample, RTT_{var} is the variance in RTT measurements, $SRTT$ is the smoothed RTT estimate. The recommended values for the constants are $alpha = \frac{1}{8}$ and $beta = \frac{1}{4}$.

Passive measurements do not induce any additional network loading because they do not inject any packets into the network. They do induce a small amount of CPU loading. However, modern processors are powerful and these processes are well within the capacity of a modern CPU at the edge network speeds that I am working with. I use passive measurement techniques to obtain the measured RTT value shown in Figure 4.3.

Active measurements are required to obtain the RTT Bloat measurement indicated in Figure 4.3 because there are no packets that normally travel the path from the Customer Provided Equipment (CPE) router to the ISP router and back. In order to obtain this measurement I must inject specially crafted packets into the network. The structure of these packets is described in detail in Chapter 6. I reduce the overhead created by injecting these packets in two ways. The first method is to reduce the size of the packet as much as possible. I use an IP packet header (20 bytes) with an 8 byte timestamp for a payload. The other way to reduce overhead is to reduce the frequency of packet injection relative to the number of data packets. This frequency controls the tradeoff between the amount of overhead created and granularity of the measurement in terms of accuracy and responsiveness. These tradeoffs are discussed in detail in Chapter 5 and Chapter 6.

Using these two measurements I calculate the RTT Natural value from Figure 4.3. RTT Natural is the the value that of the the RTT if the queues were empty or nearly so. This value is the what the delay should be and is a key value used in our BDP

Figure 4.3: Theoretical Internet Path



algorithm described in Chapter 6. The RTT Natural value is calculated as follows:

$$RTT_{Measured} = RTT_{Natural} + RTT_{Bloat} \tag{4.6}$$

$$RTT_{Natural} = RTT_{Measured} - RTT_{Bloat} \tag{4.7}$$

Where $RTT_{Measured}$ is the computed RTT using the TCP RTT calculation algorithm, RTT_{Bloat} is the queuing delay and $RTT_{Natural}$ is the RTT without any bufferbloat. Knowing the measured RTT and the bloat RTT I could (using subtraction) calculate the perfect amount of queuing time for an individual flow.

4.4 Packet Scheduling and Queue Management Algorithms

Packet scheduling and queue management are both necessary parts of the queue sizing problem. Packet schedulers separate flows into queues by classification which is necessary for good management because various flows have different characteristics and need to be managed differently. Queue management algorithms manage the length of queues. The two algorithms are often confused and it does not help that sometimes they are built into the same kernel module.[3]. However, these algorithms are separate and complimentary. Both packet scheduling and queue management should be applied.

4.4.1 Packet Scheduling Algorithms

Packet scheduling algorithms are designed to decide which packet to send next. They do nothing to manage the size of a queue. The scheduling of packets and which packet to send next is not the topic of this dissertation, however, the classification of flows is an essential part of queue sizing. Examples of packet scheduling algorithms include [8, 76, 24, 52, 7, 78]. DiffServ is a broad based framework for packet scheduling defining queues which will receive more service than others. Typical classifications are Expidited Forwarding (EF) used for VOIP and other real time flows as well as Assured Forwarding (AF) which is used for other flows. There are a variety of other DiffServ codepoints, but, EF and AF are the codepoints typically found in use on the Internet today.

The important thing from a queue sizing perspective is that packet schedulers separate flows into different queues. Typically this is done on a basis of traffic class

[3]http://man7.org/linux/man-pages/man8/tc-fq_codel.8.html

as described in the DiffServ codepoints, however, flows can be classified in many ways even individually as described in Chapter 6. This separation of flows allows individual management tailored to the unique requirements of each flow.

4.4.2 Active Queue Management (AQM)

AQM algorithms have been designed to manage a problem called bufferbloat. Bufferbloat a condition caused by excessive queuing when a flow is in a state described by Equation 4.3. Bufferbloat is the most obvious side of the queue sizing problem and results in excessive delay. The other side of the queue sizing problem is caused when the queue size is too small and the flow is in a state described by Equation 4.2. Though it is only one side of the queue sizing problem reducing bufferbloat can provide a substantial benefit to an individual flow and to the Internet in general.

There are two modern AQM algorithms that are being deployed widely though the state of activation of these algorithms is unclear as of this writing. These two algorithms are PIE and CoDel (pronounced cuddle) [69, 64]. The key advance in technology from these two algorithms is that they measure the queue size in time. This is a substantial advancement over the older algorithms such as RED and others that measured the queue size in length [22, 21, 20, 23, 48, 64, 66, 69, 93]. CoDel uses a timestamps to determine queue size. A packet is timestamped upon enqueuing and this is compared with the time the packet is dequeued to determine how long the packet has been in the queue. PIE measures the departure rate and multiplies this by the length of the queue to determine the length of the queue in time.

When CoDel detects that the queue time has been over the *threshold* parameter for an *interval* of seconds then it applies a packet marking/dropping rule. This rule decreases the mark/drop time in inverse proportion to the number of drops since the

dropping state was entered. This is meant to create a linear decrease in the rate of packets, but, is actually dependent on the RTT of a flow. When the queue time gets below the *threshold* value then CoDel resets. PIE is a little more complex. When its queue time gets below the *threshold* it calculates its drop rate according to Equation 4.11.

$$if\ p < 0.01,\ \alpha = \hat{\alpha}/8;\ \beta = \hat{\beta}/8; \tag{4.8}$$

$$if\ p < 0.10,\ \alpha = \hat{\alpha}/2;\ \beta = \hat{\beta}/2; \tag{4.9}$$

$$if\ p < 1,\ \alpha = \hat{\alpha};\ \beta = \hat{\beta}; \tag{4.10}$$

$$p = p + \alpha * (cur\_del - ref\_del) + \beta * (cur\_del - old\_del); \tag{4.11}$$

Where α and β are tuning parameters, *cur_del* is the currently calculated delay and *old_delay* is the calculated delay from the last drop time. Equations 4.8, 4.9 and 4.10 are for auto tuning the drop rate.

There are two problems with these algorithms. First the algorithms measure queue size directly and cannot queuing problems that occur in other queues. This shortcoming is discussed in detail in Chapter 5. The second problem is that they cannot manage flows with a large queuing delay. Both algorithms suffer significant loss of throughput as described by Equation 4.2 as the queuing delay increases beyond 250 ms. as discussed in Chapter 6. The problems with these two algorithms are caused by the fact that they directly measure queue length on a single queue rather than across a link and that they cannot adapt to RTT flows.

4.5 Conclusions

The collection of networking knowledge in this Chapter allows us to discern the big picture and understand the problem of "bufferbloat" or queue sizing. The queuing theory section describes the equations needed to determine the individual queue size for each flow according to its bandwidth delay product. The networking topology section tells us that occurs at the slowest link which is nearly always at the edge. In addition, it shows us that the queuing often occurs within the link below the IP layer. This effect is described in detail in Chapter 5. The Transport section describes the Interaction of TCPs with the network queue size and the network sensing chapter describes how the parameters necessary to operate the bandwidth delay product equations can be obtained. Packet scheduling is a sister science of queue sizing that is often mistakenly thought of as a replacement for queue sizing. Packet scheduling determines which packet to send next while queue sizing determines how many packets to hold in the queue. These two types of algorithm should be operated together. Finally the AQM section describes the state of the art in queue management and the shortcomings of the current generation of algorithms that we address in Chapters 5 and 6.

Chapter 5

The Active Sense Queue Management (ASQM) Algorithm

5.1 Introduction

Active Queue Management (AQM) algorithms have seen a lot of attention in recent academic literature as well as in the popular press. The problem with traditional AQM is that it is designed to operate on queues at the IP layer, which is not always where the problem commonly called "bufferbloat" occurs. Bufferbloat can move about among many queues some of which are resistant to traditional AQM such as Layer 2 MAC protocols used in cable/DSL links. I call this problem bufferbloat displacement. I discussed the causes of the bufferbloat displacement problem in detail in Chapter 4. Our contribution is a new class of AQM algorithm called Active Sense Queue Management (ASQM). ASQM is an IP layer AQM protocol that uses active sensing techniques to gain an understanding of the queuing topology in the network neighborhood. ASQM uses this additional knowledge of RTTs to manage bufferbloat in neighboring queues.

In the chapter I describe how ASQM is used to manage Layer 2 link delay that occurs in an ISP access link. This link delay is commonly known as primary bufferbloat. In this dissertation I describe how to expand the ASQM configuration to help manage

secondary bufferbloat. The typical causes of secondary bufferbloat are business disputes where one Autonomous System (AS) uses packet scheduling to slow down traffic from another AS or provider. Examples of these Advanced DiffServ (Differentiated Services) packet scheduling algorithms are found in [8, 76, 24, 52, 7, 78]. The packet scheduling causes a secondary bottleneck in the flow of data causing secondary bufferbloat. ASQM can be configured to manage either type of bufferbloat. I provide testbed experiments comparing ASQM's throughput delay characteristics with traditional AQM algorithms demonstrating ASQM's ability to manage bufferbloat displacement where traditional AQM algorithms cannot.

The Federal Communications Commission (FCC) produces a report every year called Measuring Broadband America.[1] In this report from the years 2010 through 2014 the 80/80 study designed by the North Carolina State University Institute for Advanced Analytics determines how often the access link does not meet the ISP committed rate at the IP layer in the modem during peak traffic hours (from 7:00 pm to 11:00 pm). In 2014 50% of the 16 ISPs measured provided 90% of the committed rate to 80% of the panelists 80% of the time. The other 50% of ISPs provided less than 90% of the committed rate. These detailed studies conducted over years demonstrate that bufferbloat displacement into the ISP access link is a serious problem.

The queue management problem commonly called bufferbloat has been with us for a long time. It is caused by an interaction between the ubiquitous TCP algorithm and queue engineering in the path. Typical TCP algorithms are loss based. Examples of loss based algorithms come in many flavors with varying levels of aggression [17, 65, 61, 10, 56, 25]. Algorithms commonly deployed and active on the Internet today include Cubic (Linux), Compound (Windows XP), and NewReno (mac, Windows Vista+), [19,

[1] http://data.fcc.gov/download/measuring-broadband-america/2014/2014-Fixed-Measuring-Broadband-America-Report.pdf

68, 80]. Loss based TCP protocols seek to fill the slowest queue as much as possible and increase bufferbloat. The ubiquitously deployed TCP Cubic and NewReno are loss based. TCP Compound is a hybrid of loss and delay based TCP.

Recent increases in the range of throughput capabilities found in network devices have made the problem more exigent. A typical cable modem in use today provides throughput from 2-50 Mbps. Statically engineering queue size for a device such as this is essentially impossible because the throughput and delay characteristics vary too much. A queue engineered for a delay of 100 ms at 50 Mbps would be 25 times too large at 2 Mbps producing 2.5 seconds worth of delay. Faster speeds are leading to larger queue sizes making the problem worse for lesser speeds.

Delay based TCP protocols are sensitive to RTT and eliminate bufferbloat except when in the presence of a loss based TCP such as Cubic or NewReno. This makes delay based TCP protocols ineffective against bufferbloat in general because they are likely to face one of the ubiquitously deployed loss based NewReno or Cubic TCPs. However, in a controlled environment delay based TCP can reduce bufferbloat. Examples of delay based TCP are Vegas, CAIA CDG, and Fast, [36, 82, 9]. Many other TCP variants have been created over the years including equation based TCP [33], Multipath TCP [34], split TCP [41], and Network Coding TCP (NCTP) [79]. However, no TCP variant exists today that can control bufferbloat in the presence of the ubiquitously deployed loss based Cubic and NewReno TCPs.

Explicit Congestion Notification (ECN) enhances the performance of AQM by marking packets rather than dropping them. Loss free data transmission is possible using ECN because the TCP server (sender) reacts to marked packets as if they were dropped packets [32, 72]. The state of deployment and activation of ECN in the Internet is poor but improving, about 40 percent at the core as of this writing [4, 51]. Some middleboxes (proxies) still do not duplicate these options properly, [37]. If ECN

is deployed then the AQM should use ECN marking instead of dropping. However, an AQM algorithm must be able to drop packets as a fallback position.

There are two dynamic AQM algorithms deployed today to manage queue size and control bufferbloat: CoDel and PIE, [64, 69]. The weakness of these algorithms is that they do not control the bufferbloat problem in queues below the IP layer. I call this problem bufferbloat displacement. Queuing theory tells us that bufferbloat is displaced into slowest queue(s) in the path. When the ISP access link cannot meet the IP layer ISP committed rate then the bufferbloat is displaced into the access link filling the MAC layer queues and causing bufferbloat that CoDel and PIE cannot detect.

AQM algorithms are designed to manage bufferbloat. The IETF currently recommends that an AQM algorithm should be implemented in network devices in compliment with a DiffServ scheduler.[2] Recent and popular AQM algorithms such as CoDel and PIE use burst size (queue delay over time) to control the queue, [64, 69]. Older algorithms such as RED and its variants measure queue size directly, [23, 22, 48, 21].

The problem with traditional AQM is that it does not protect Layer 2 MAC protocols from bufferbloat. These protocols are resistant to traditional end to end AQM because they are Link Layer protocols and do not communicate with the TCP sender. In ASQM I take a different approach using active sensing at the IP Layer to determine the queuing delay across the Link Layer and send a mark/drop signal to the end to end TCP sender to slow down.

The FCC study shows that a significant amount of bufferbloat occurs during peak traffic hours that traditional AQM cannot detect because the access link is providing less than the ISP committed rate. So I develop ASQM to solve this problem. ASQM uses an active sensing mechanism to detect bufferbloat across a neighboring link. Our ASQM algorithm runs at the IP layer, but, senses queuing delay across the entire

[2]http://tools.ietf.org/html/draft-ietf-aqm-recommendation-03

neighboring link. This novel approach allows ASQM to remove bufferbloat from a neighboring link such as the ISP access link 100% of the time even during peak traffic hours when bufferbloat displacement occurs that CoDel and PIE cannot detect.

I believe that a different approach is required. It is unreasonable to expect all ISPs to maintain 100 percent of the advertised throughput 100 percent of the time. ISPs are faced with a wide variety of conditions under which they must operate. Distance, competing traffic, and RF conditions on the wire (especially for twisted pair) affect their networks and 90 percent of advertised rates during peak periods is quite reasonable performance. I propose ASQM as an answer to the problem. ASQM uses active sensing between the IP layers to detect bufferbloat and manage all of the queues between them regardless of where in the link the bufferbloat is occuring.

ASQM uses an active sensing approach. Active sensing creates a trade-off between overhead (from the sensory packets), measurement accuracy and feasibility. Our contribution in this Chapter is to investigate these trade-offs and provide the most feasibility and accuracy with the least amount of overhead. Our ASQM algorithm controls bufferbloat on par with CoDel and PIE during non-peak traffic hours and continues to provide excellent bufferbloat protection during peak traffic hours when CoDel and PIE cannot. Our ASQM algorithm can be deployed on any ISP modem or router and requires less than 0.5% overhead.

5.2 Active Sense Queue Management

Our ASQM algorithm uses active sense packets to determine the queuing delay in the access link. Successfully employing an active sensory system requires a detailed understanding of the queuing arrangements in the layers below the IP as well as the DiffServ packet scheduling applied by ISPs. In addition active systems create overhead

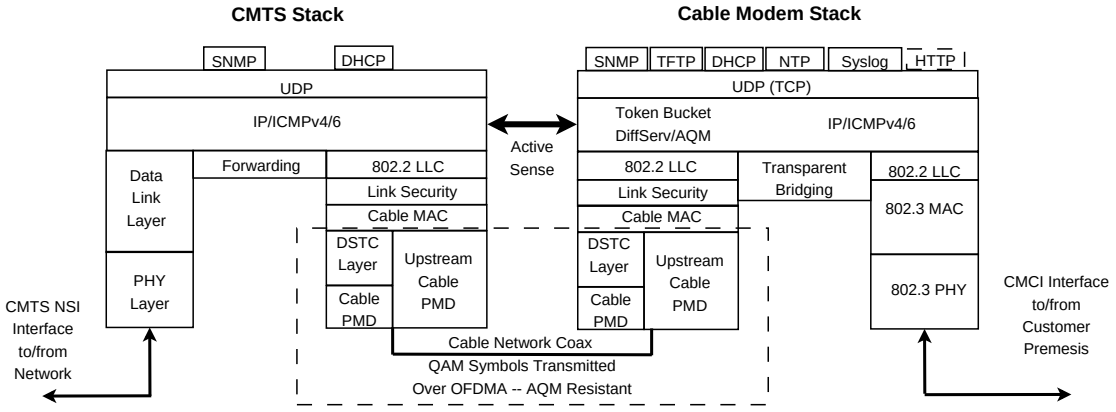


Figure 5.1: Primary Bottleneck – Cable Modem Termination System to Cable Modem Link (DOCSIS 3.1)

so careful attention must be taken in order to reduce overhead.

Figure 5.1 shows a detail of the CMTS and modem stacks. DSL technology uses a similarly arranged stack. Packets flowing through the system in either direction (upload/download) encounter a hierarchical token bucket rate limiter used by the ISP located at IP in Layer 3. This is the ISP rate limiter where AQM will be deployed.

As demonstrated in Nichols and Jacobson any queue in the lower layers that is slower than the advertised (token bucket) rate will defeat the DiffServ/AQM system at the IP layer [64]. The FCC study Measuring Broadband America indicates that this happens frequently especially during peak traffic periods from 7:00 pm to 11:00 pm. This portion of the link is called out in Figure 5.1 by a dashed rectangle. I did not include the 802.2 LLC since these layers are extremely fast and unlikely to suffer from bufferbloat. In any case some form of traditional AQM is still possible at the 802.2 LLC layer.

At the cable MAC, AQM activities become impossible without a significant redesign of the protocol and a re-assignment of layering responsibilities. AQM protocols must be able to drop packets. Dropping frames is in direct conflict with the MAC's retransmission scheme. ECN marking is preferable to dropping, however, in cases where

ECN is not implemented in every device in the path dropping is required. Even in cases where ECN is implemented in every device, it is impossible to guarantee that the proper bits exist in a MAC frame to signal an ECN mark because fragmentation could have already occurred.

At layers below the cable MAC, dropping or otherwise signaling the sending end to slow down is even more improbably difficult. The Discrete Space Time Coder (DSTC) converts MAC frames into QAM signals for transmission by the Physical Media Dependent layer across the wire. At this point concepts such as packets and end to end signaling no longer exist and traditional AQM activities are impossible. ASQM is our answer to these problems. ASQM is an IP layer protocol that uses active sensing to determine delay occurring in all of the queues in the lower layers as shown in Figure 5.1. The active sensing mechanism consists of packets sent from IP Layer to IP Layer.

ASQM's sensing packets have been the subject of much discussion and research in our laboratory. The goal is to measure the RTT of the access link. I have tried ICMP packets, SNMP packets and injected packets as well as periodic sensory packets and sensory packets that are proportionate to the flow. Each method offers trade-offs between overhead, sensing accuracy, scalability and feasibility. For the sake of brevity I only discuss sensory packet injection in proportion to the flow in detail.

Our ASQM algorithm generates sensory packets by copying a packet header from the flow, manipulating the IP address fields, marking, timestamping and injecting them into the stack. These sensory packets flow from the IP layer at the modem (where they are created) to the IP layer at the ISP router and back. Upon receiving a returning sensory packet our ASQM algorithm generates link RTT samples by subtracting the timestamp from the current time. Then our ASQM algorithm generates a link RTT estimate by smoothing the estimates according to the calculation detailed in RFC

6928.[3]

Many (if not all) ISPs employ DiffServ packet scheduling (not a substitue for AQM) in order to separate voip and other high priority real time traffic from best effort traffic. The high priority traffic is forwarded into the Expedited Forwarding (EF) queue and the best effort traffic goes into the Assured Forwarding queue (AF). The EF queue serves low throughput flows such as voip and needs no AQM. ASQM operates only on the AF queue for best effort flows. This assures that sensory packets transmitted across the access link always go through the correct queuing.

The next step is to find ways to reduce overhead. There are two ways of doing this; reduce the size of the sensory packet and reducing the ratio of sensory packets relative to data packets. ASQM uses a 20 byte IP packet header with an 8 byte timestamp. This is the smallest packet I could design and transmitted at a ratio of one sensory packet for every four data packets this translates into an overhead of less than 0.5% when using 1500 byte packets and even less when using larger MTUs. These packets are forwarded using an *iptables* rule.[4] Smoothing is accomplished using the TCP RTT calculation.

In our design of ASQM's sensory packets I have taken a great deal of care to reduce overhead and to ensure that the packets flow through the correct queue at the ISP. In any case I expect the ISP to cooperate or at least not actively seek to confuse the sensory packets. With this cooperation in mind I can take a very accurate measurement of queuing delay encountered in the access link. The next step is to design our ASQM algorithm to take corrective action (marking/dropping) when bufferbloat is detected in the access link.

When the smoothed link estimate reaches a threshold value that I call *target* (de-

[3]https://tools.ietf.org/html/rfc6298
[4]http://linux.die.net/man/8/iptables

fault 100 ms) then our ASQM algorithm takes corrective action. The 100 ms target queue size is a default tunable parameter. The default target queue size value was chosen based on our own experiments and on the default value from Nichols and Jacobson [64]. This default value has been shown to give good performance for a wide range of RTTs from 10 ms to 500 ms. ASQM's dropping/marking activities are governed by a control law that produces an approximately linear slowdown from the sender. Starting with an interval of 100 ms (default) each drop interval is reduced by $1/\sqrt{n}$ where n is the number of marks/drops since the beginning of marking/dropping activity. When an active sense measurement less than 100 ms is received marking/dropping activity ceases and n is reset to zero.

ASQM's active sensing mechanism measures round trip queuing delay in the access link and the algorithm takes corrective action (marking/dropping) when the queuing delay exceeds the target threshold. ASQM is designed to be employed in both the upload and download direction on the best effort (AF) DiffServ queue. Using this novel mechanism ASQM is able to remove bufferbloat from the access link even during peak traffic hours when other algorithms are unable to detect the bufferbloat.

5.3 Evaluation methodology and testbed

The evaluation I undertook tried to illuminate ASQMs queue management capabilities by comparing ASQM to popular AQM algorithms such as CoDel and PIE [64, 69]. The goal of this evaluation was to demonstrate the throughput delay tradeoffs of each algorithm both when the Layer 2 link was providing 100% of the rated speed and when the link is only providing 90% because of traffic. All three algorithms performed well

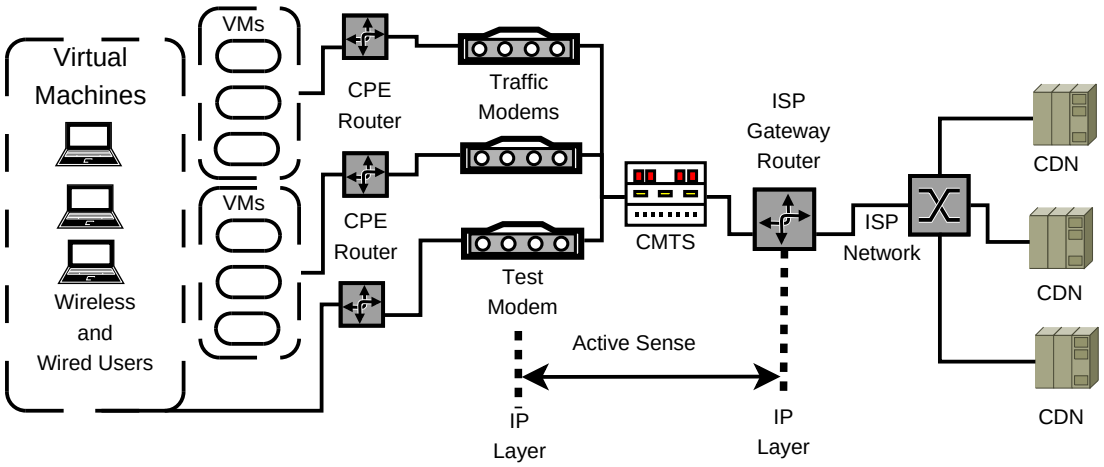


Figure 5.2: Hardware emulation testbed

when the Layer 2 link was providing 100% of the advertised speed. However, ASQM continued to perform well even when the Layer 2 link was only providing 90% of the advertised speed as is common during peak usage hours.

The testbed I constructed is shown in Figure 5.2. The traffic was generated by virtual user machines and CDNs constructed from PCs running the Linux 3.2 kernel. The traffic was generated as specified in the IETF draft AQM Evaluation Guidelines; five repeating TCP transfers of 5MB each, one continuous TCP transfer and four HTTP web traffic (repeated downloads of 700kB).[5] This traffic configuration is specifically designed to investigate bufferbloat particularly with a mix of short flows in combination with long flows.

Experiments were performed in the upload (data flowing from the users to the CDNs) as well as in the download direction. Each experiment was run for 120 seconds with 5 experimental runs then compiled into throughput and delay CDFs. throughput and delay CDFs were provided for all three algorithms in each link speed configuration (100% and 90% of the committed speed). The experiments cover a range of RTTs from 50 ms to 200 ms because this range represents in large part the conditions that will be

[5]http://tools.ietf.org/html/draft-kuhn-aqm-eval-guidelines-00#section-3.2.4

found in end user access links. In any case all three of the algorithms (CoDel, PIE and our ASQM) begin to break down above 250 ms and become unusable by 500 ms.

The CPE routers, the modems, the CMTS and the ISP gateway were constructed from PCs running the Linux 3.15 kernel. The modems and CMTS were equipped with ASQM, CoDel and PIE, [64, 69]. This was done using a Hierarchical Token Bucket (HTB) as in common in routers from Cisco and other manufacturers.[6] I note that the HTB is a packet scheduler and does not manage queue size.

5.4 Evaluation

The goal of this evaluation was first to demonstrate that ASQM performs on par with CoDel and PIE during non-peak usage hours. All AQM algorithms are expected to perform well when the link is providing 100% or more of the rated throughput. Secondly I wanted the evaluation to show that ASQM is the only algorithm that continues to perform well when the link is providing less than 100% of the rated throughput as is common during peak usage hours.

I have performed thousands of experiments with a large range of network factors and parameters; RTT 10-1000 ms, throughput 1-50 Mbps, target queue size 10-500 ms in the upload and download directions. In order to demonstrate that ASQM achieves both goals I have chosen to present two sets of graphs for each algorithm. I have chosen to present two sets of CDFs for each algorithm demonstrating AQM behavior during non-peak hours (100% committed rate or more) and during peak hours (less than 100% committed rate)

The modems in both sets of experiments were configured to provide 8 Mbps committed rate and 16 Mbps peak rate (using HTB borrowing). For the non-peak traffic hours

[6]http://linux.die.net/man/8/tc-htb

set the link up to provide the peak rate for each modem simultaneously (48 Mbps). With this configuration I expected to see about 15.5 Mbps from each modem. For the peak traffic experiments I set the link up to provide about 90% of the committed rate for each of the three modems (21 Mbps). With this configuration I expected to see about 6.5 Mbps from each modem.

The CDF's presented examine the throughput and delay characteristics of each algorithm in each traffic condition. I found that all algorithms perform well in terms of throughput in both peak and non- peak traffic scenarios. In terms of RTT all algorithms perform well in non-peak traffic scenarios, but, only ASQM performs well when buffering displacement occurs because of peak traffic conditions.

5.4.1 AQM During Non-Peak Hours (100% Committed Rate or More)

I present this series of CDFs in order to demonstrate that our ASQM algorithm performs on par with CoDel and PIE during non-peak traffic hours. In Figure 5.3, I present the end to end delay curves for a CoDel (with default parameters) managed link. I examined three different RTTs from 50 ms to 200 ms. CoDel had excellent RTT response across the range of RTTs. At 50 ms RTT CoDel's management kept the end to end delay within a range from 50-75 ms. At 100 ms RTT the end to end delay range was 100-125 ms and at 200 ms RTT the range was 200-225 ms. In Figure 5.4, I present corresponding throughput curves for these experiments.

Each link had a committed rate of 8 Mbps with a peak rate of 16 Mbps. Since the the non-peak hours link had enough throughput to supply all three modems with their full peak rate each modem was able to develop about 15.5 Mbps measured throughput. They did not reach 16 Mbps because of overhead from Ethernet, IP and Transport

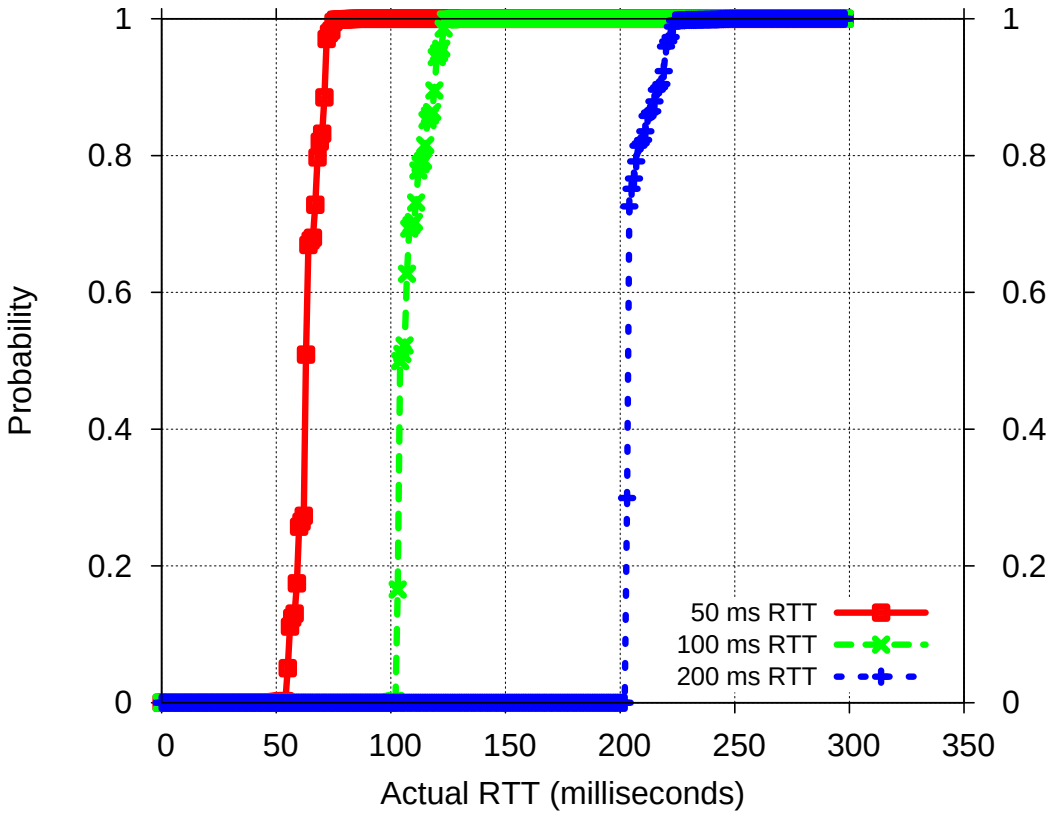


Figure 5.3: CoDel RTT CDF (Non-Peak Hours)

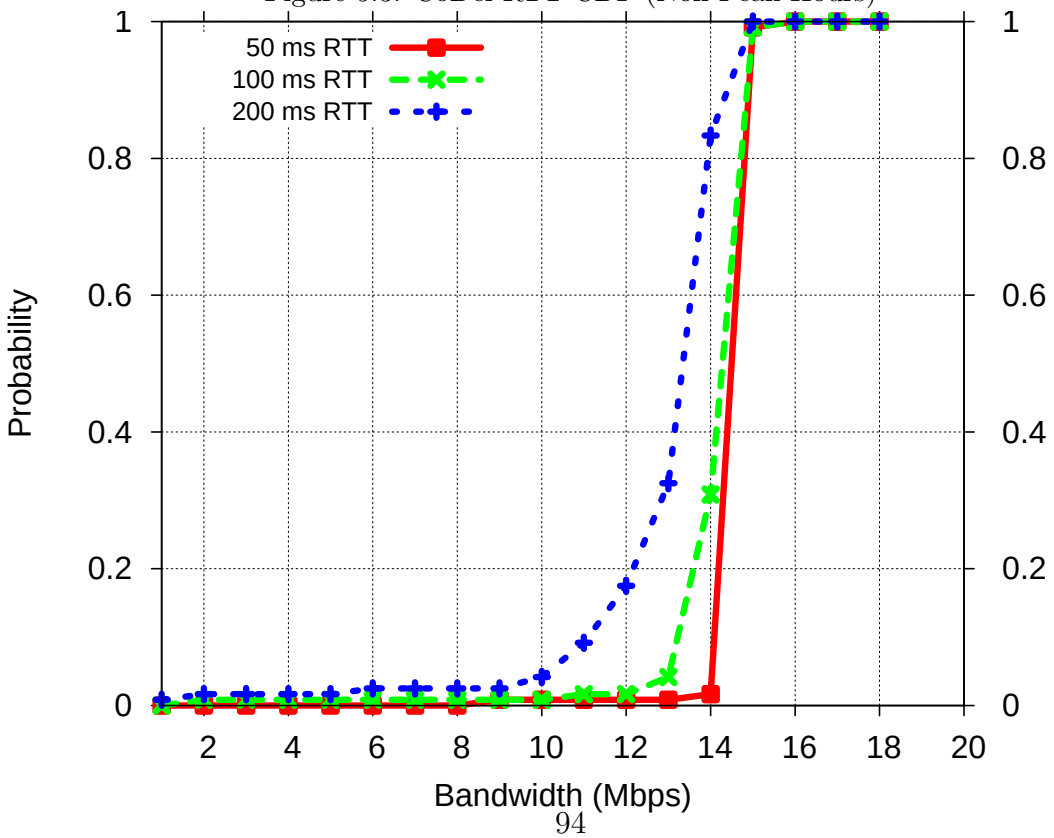


Figure 5.4: CoDel Throughput CDF (Non-Peak Hours)

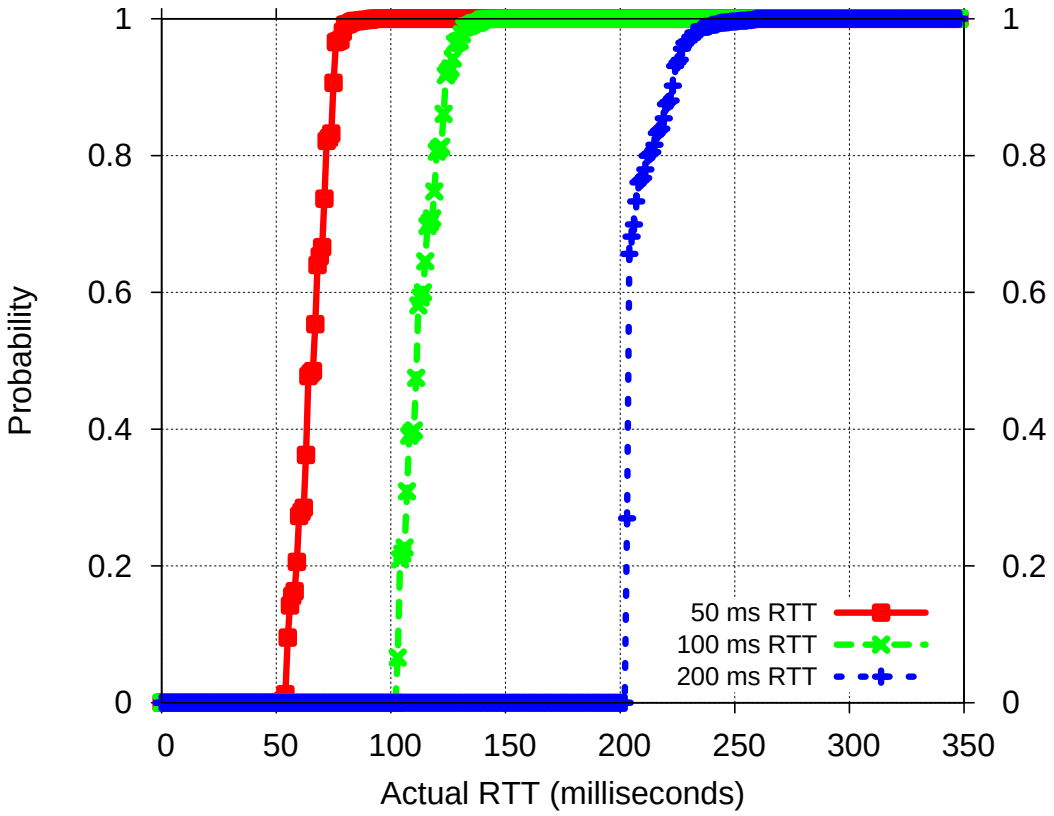


Figure 5.5: PIE RTT CDF (Non-Peak Hours)

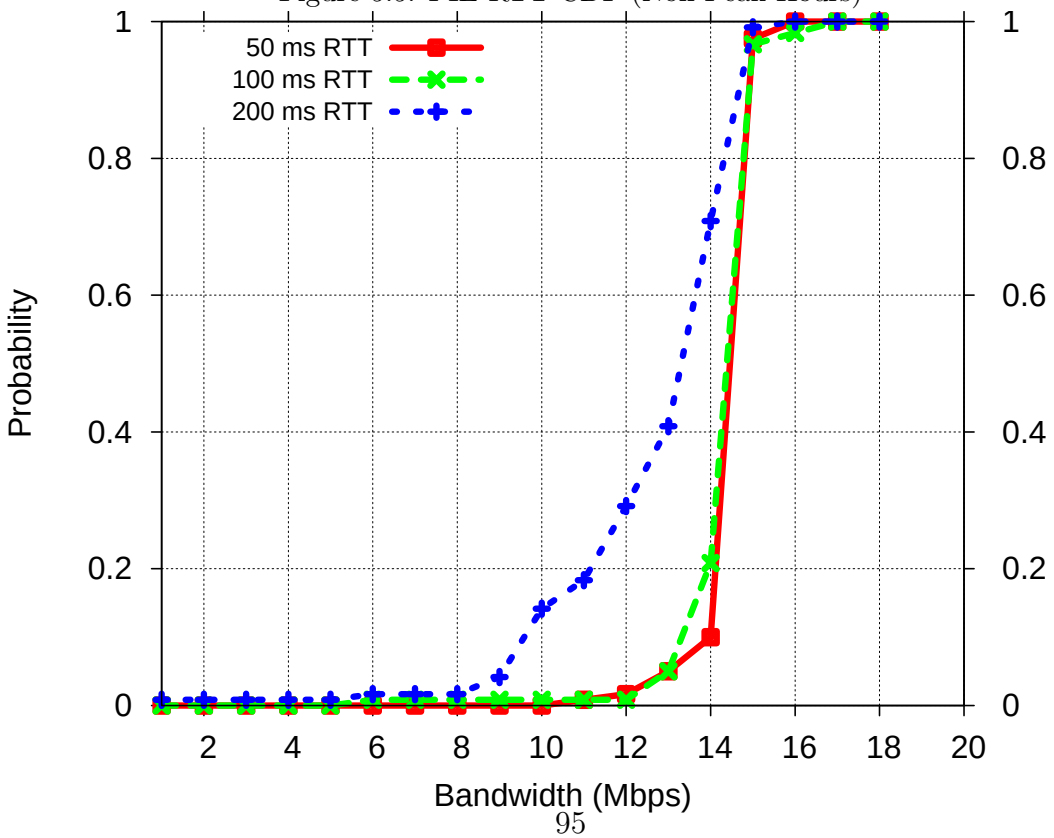


Figure 5.6: PIE Throughput CDF (Non-Peak Hours)

headers. When the RTT was 50 ms the CoDel managed link developed full throughput. However as the RTT increased the throughput decreased slightly. This is because CoDel was keeping the queue size at about 100 ms (the default) which is slightly too small for the 200 ms flow.

In Figure 5.5, I present the end to end delay curves for a PIE managed link across a range of RTTs (50-200 ms). PIE also had an excellent RTT response. It kept the actual RTT range experienced by the link to about 50-75 ms for a 50 ms RTT link, 100-125 ms for a 100 ms RTT link and 200-225 ms for a 200 ms link for 90% of the measurements.

In Figure 5.6, I present the throughput curves for PIE. The CDF curves show that PIE delivered similar throughput performance as CoDel at 50 and 100 ms RTT. However, PIE delivered slightly less throughput than CoDel at 200 ms RTT. This shows that PIE was slightly more aggressive with its dropping policy. In any case, both of these algorithms deliver excellent queue management characteristics across a wide range of RTTs.

In Figures 5.7 and 5.8, I present the throughput and delay curves for ASQM. ASQM also delivered excellent RTT response as shown in Figure 5.7 (although not quite as good as CoDel and PIE). ASQM allowed the actual measured RTT to exceed the link RTT by only as much as 50 ms. In Figure 5.8, I present the throughput curves for ASQM. ASQM acheived slightly more throughput than either CoDel or PIE across the range of link RTTs (50-200 ms). This is because ASQM is slightly less aggressive than CoDel or PIE in its dropping policy.

These throughput and delay CDFs show that all three algorithms are fully capable of delivering excellent queue management during non-peak traffic hours when the link is capable of providing 100% or more of the rated throughput. Each algorithm performs slightly better or worse than the others in given scenarios. These slight differences are

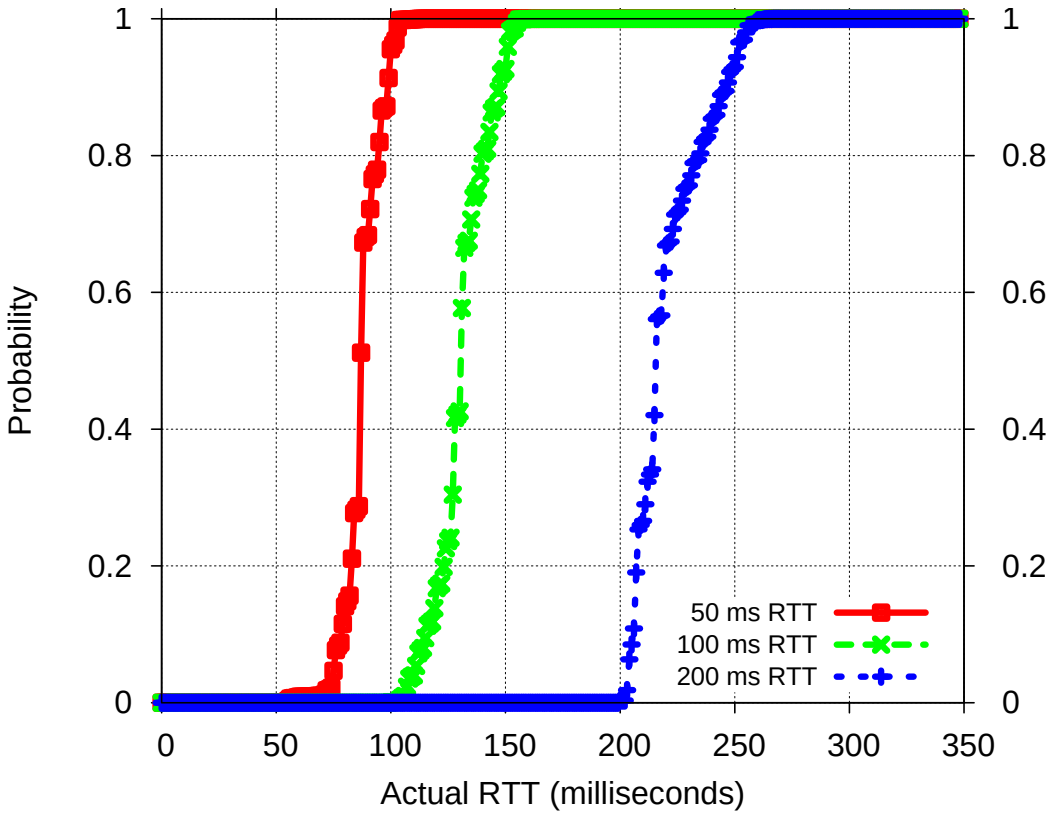


Figure 5.7: ASQM RTT CDF (Non-Peak Hours)

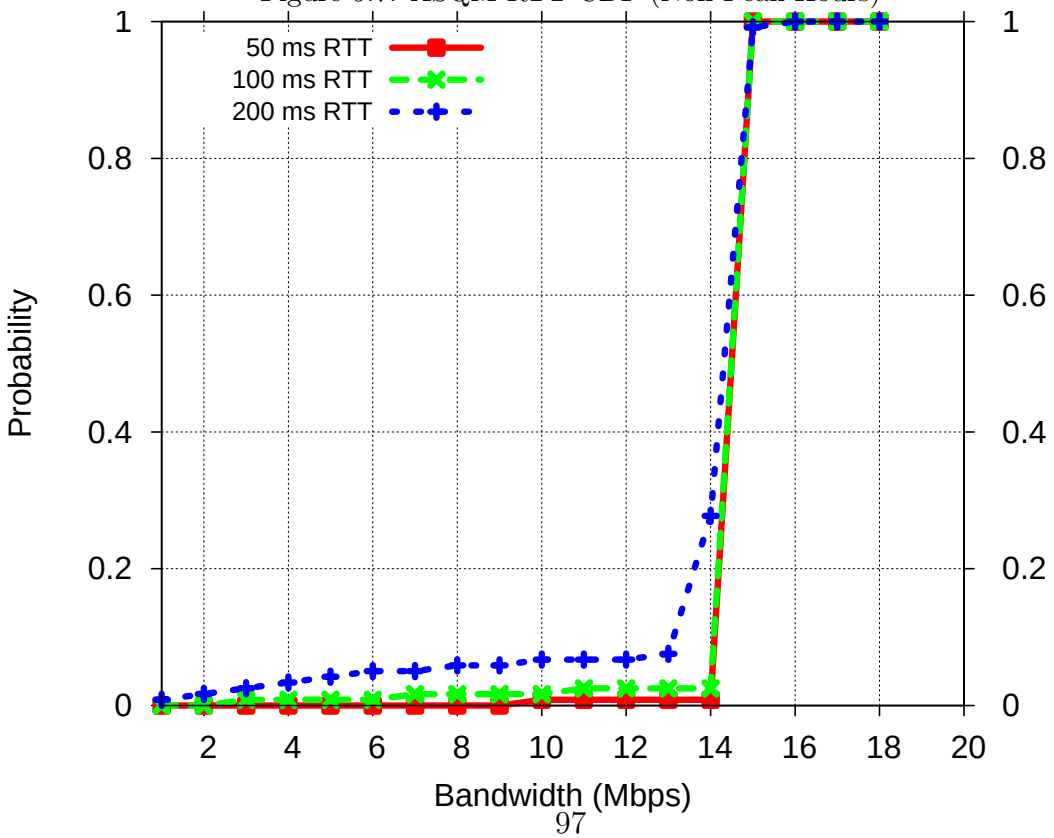


Figure 5.8: ASQM Throughput CDF (Non-Peak Hours)

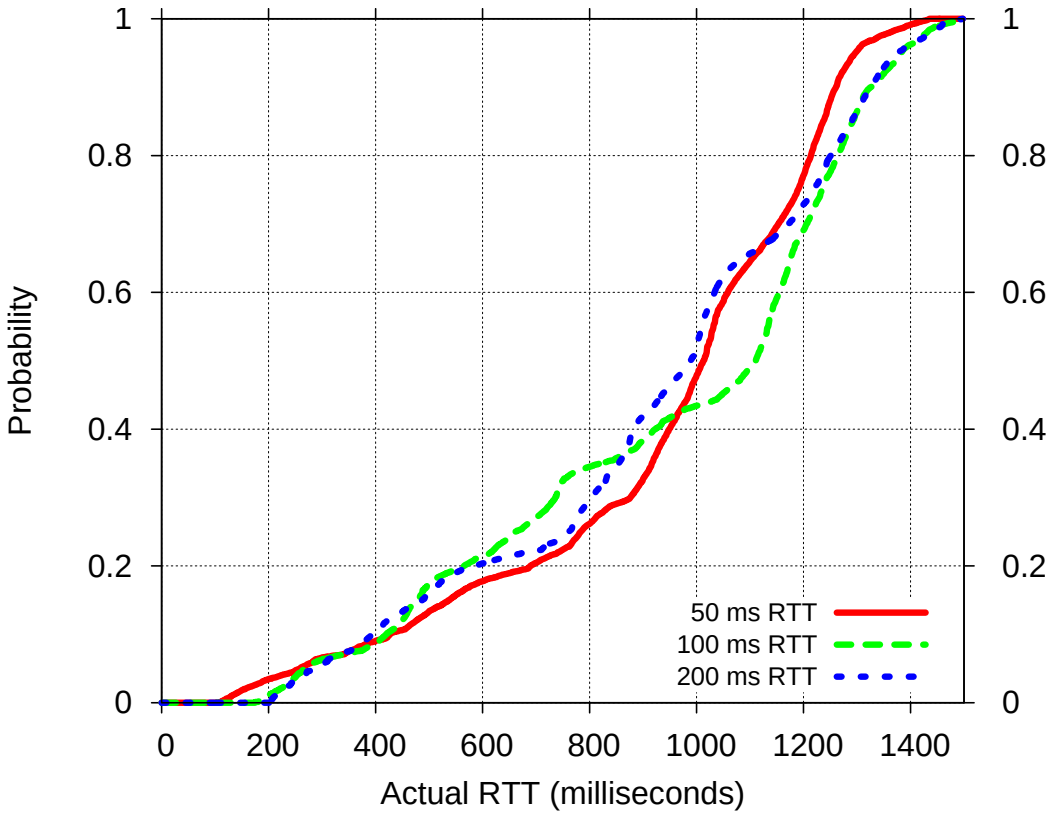


Figure 5.9: CoDel RTT CDF (Peak Hours)

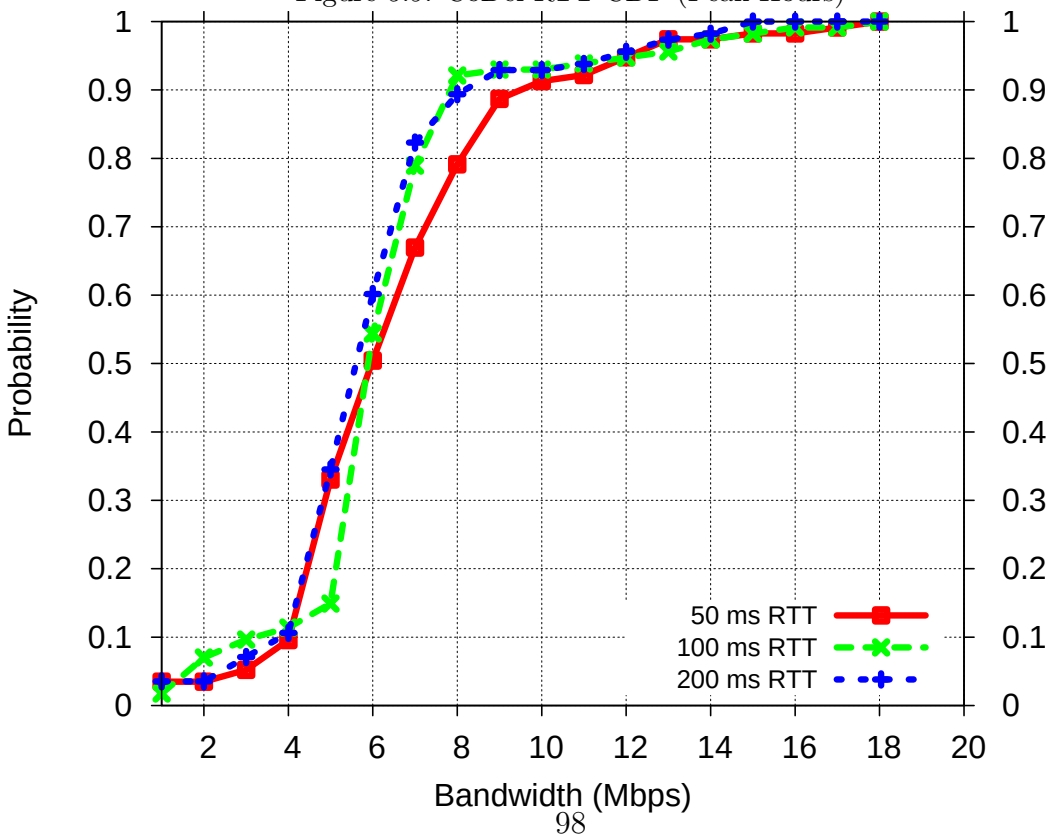


Figure 5.10: CoDel Throughput CDF (Peak Hours)

insignificant to the end user. As expected each of these three algorithms performed well in the non-peak traffic scenario and ASQM is on par with the others.

5.4.2 AQM During Peak Hours (Less than 100% Committed Rate)

I present this series of CDFs in order to demonstrate that our ASQM algorithm performs continues to perform excellently during peak traffic hours when CoDel and PIE cannot. In order to demonstrate this performance I designed a series of experiments designed to emulate peak traffic periods in ISP networks. All three modems had a committed rate of 8 Mbps and a peak rate of 16 Mbps. The access link however was only capable of delivering 21 Mbps (about 90% of the committed rates for all three modems) as is common during peak traffic hours from 7:00pm to 11:00pm. This caused bufferbloat displacement defeating PIE and CoDel's ability to manage the queue size. ASQM was able to manage the queue size regardless of the bufferbloat displacement.

In Figures 5.9 and 5.10, I present the throughput delay curves for CoDel. Figure 5.9 shows that CoDel was unable to manage the queuing delay in this scenario. The actual RTTs varied widely from 100 ms to about 1500 ms. This is because bufferbloat displacement caused the queuing delay to move into the link where CoDel cannot detect it. Even though the real RTT had skyrocketed to 1500 ms CoDel still did not take corrective action.

Figure 5.10 shows the throughput curves for CoDel. The throughput is a lot more variable than in the non-peak traffic hours experiments where CoDel was able to control the queue. This is symptomatic of an uncontrolled queue. Each modem builds up the queue until full and then a massive number of packets are dropped causing the throughput of that particular modem to crash. When this happens the other two

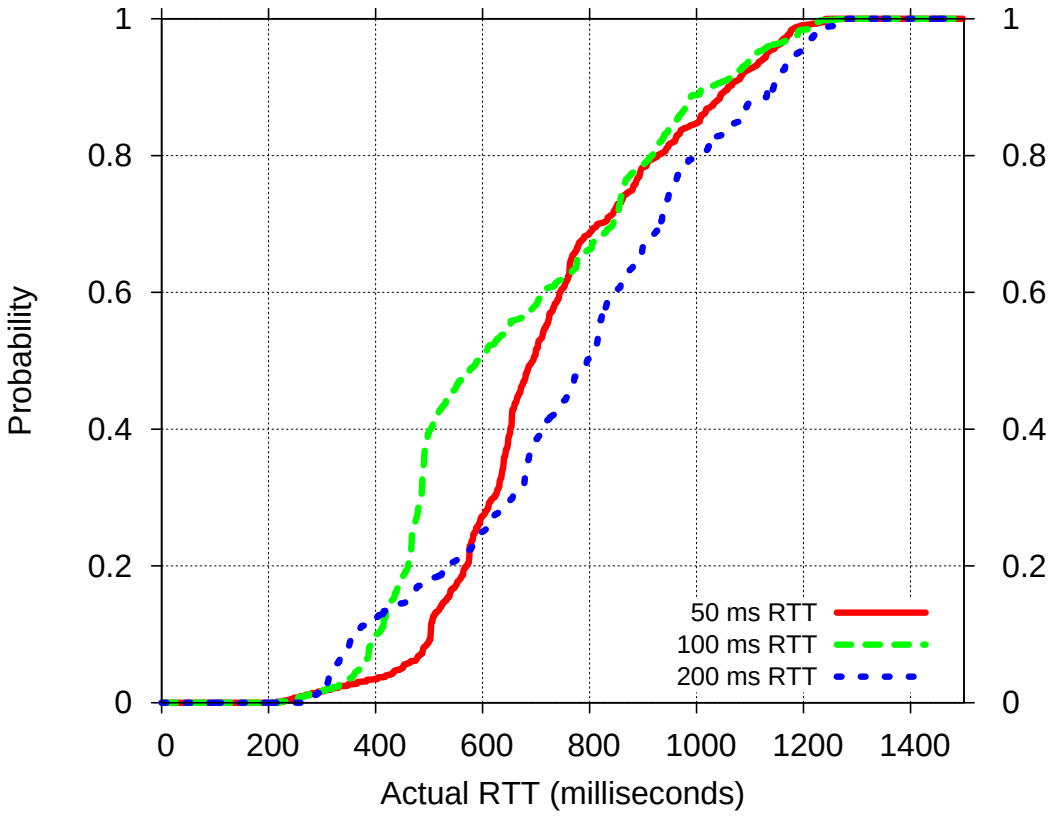


Figure 5.11: PIE RTT CDF (Peak Hours)

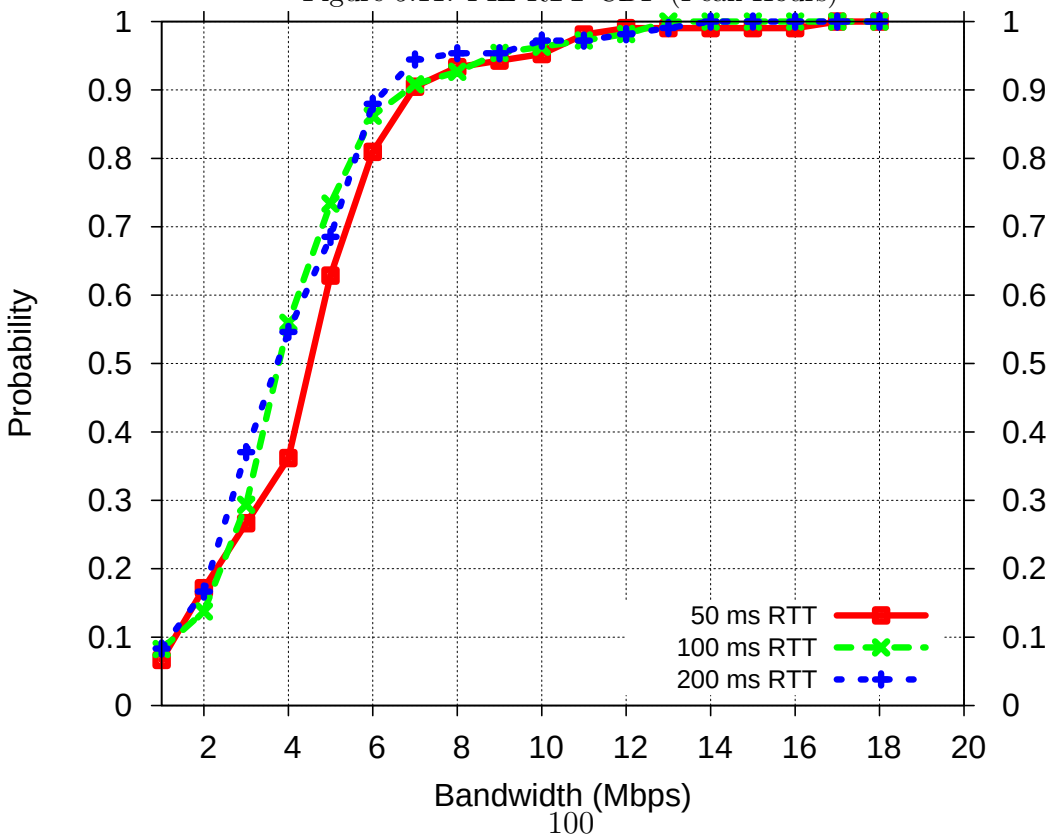


Figure 5.12: PIE Throughput CDF (Peak Hours)

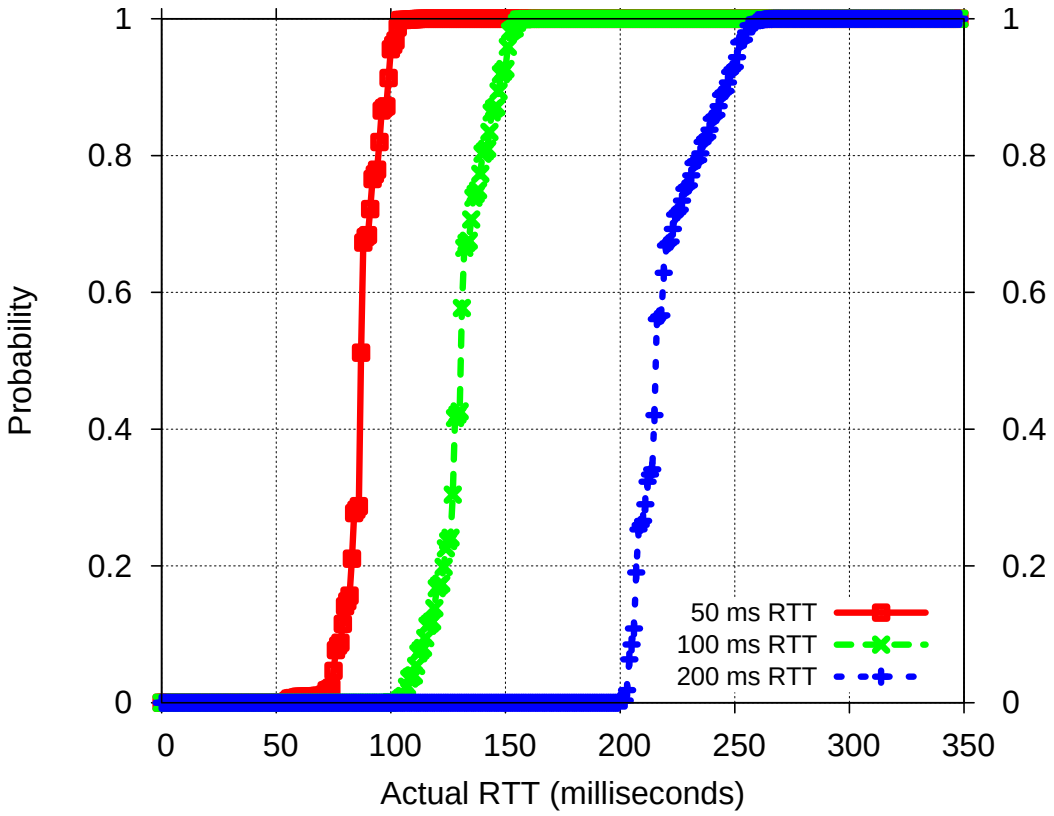


Figure 5.13: ASQM RTT CDF (Peak Hours)

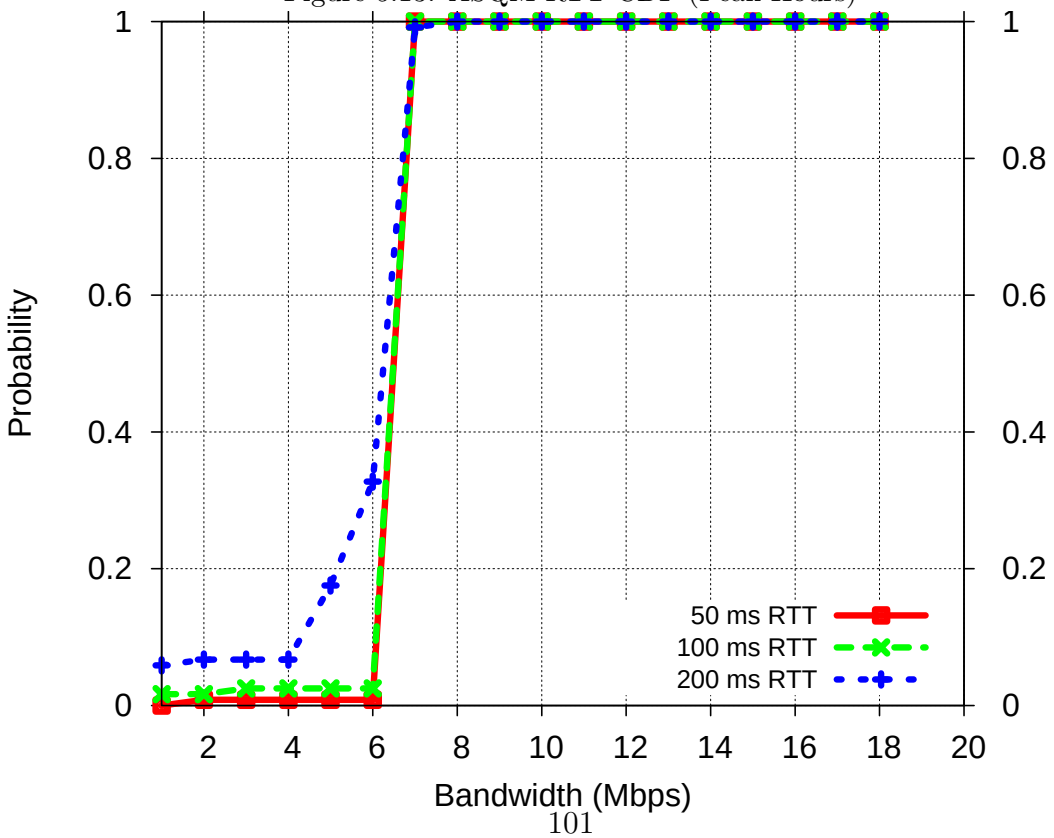


Figure 5.14: ASQM Throughput CDF (Peak Hours

modems grab the free throughput allowing their throughput to temporarily surge.

Figures 5.11 and 5.12 show a similar story for PIE. The actual measured RTTs varied widely from about 200 ms to about 1200 ms. Like CoDel, PIE was unable to manage the delay effectively when buffering displacement occurred. Actual RTT reached 1200 ms without PIE taking corrective action. I present the throughput curves for PIE in Figure 5.12. Like the throughput curves for CoDel during peak traffic hours PIE's throughput curves had a lot of variability. The cause of this variability is that PIE's control mechanism is not taking corrective action to control the queue.

Figures 5.13 and 5.14 show the throughput delay CDFs for ASQM during peak traffic hours. ASQM's active sensing mechanism detected queuing delay across the link regardless of the buffering displacement. Because of this ASQM took corrective action when the queuing delay exceeded its target value. Figure 5.13 I see that ASQM's delay curves were virtually unaffected by the buffering displacement. Figure 5.14 shows the throughput curves for ASQM during peak traffic hours. The throughput is stable at the full available rate because ASQM was controlling the queuing delay.

This evaluation fulfilled both of its goals. I have shown that all three algorithms perform well in terms of throughput and RTT during non-peak traffic hours when the access link provides 100% or more of the rated throughput. Additionally I have shown that during peak traffic hours when bufferbloat displacement commonly occurs that the other algorithms are unable to control RTT. Of the three only ASQM is able to provide a stable throughput at the full available rate with a managed RTT when bufferbloat displacement caused by peak traffic hours occurs.

5.5 Summary, Conclusions and Future Work

In this Chapter I have presented ASQM. ASQM is a new class of AQM algorithm using an active sensing mechanism to detect queuing delay across the entire access link. Traditional AQM algorithms can only detect queuing delay in the IP layer. I have presented experiments demonstrating how ASQM is able to manage queuing delay even when bufferbloat displacement occurs during peak traffic hours.

I have conducted thousands of experiments (besides the few presented in this chapter). I have found that it is irrelevant how much less than 100% of the committed rate is provided. Even the slightest bit less than 100% causes bufferbloat displacement and defeats PIE and CoDel's ability to sense bufferbloat. I chose to present a range of RTTs from 50 ms to 200 ms. I chose these values because they provide a good range around the 100 ms target delay. Also I wanted to avoid the loss of throughput that occurs in all algorithms at larger delays. This is a well known problem in the AQM field called the Long Delay Flow problem.

PIE and CoDel both purport an operating range of 10 ms to 500 ms, ASQM's operating range is about the same. However, all three algorithms begin losing throughput due to the Long Delay Flow problem at around 250 ms. After 500 ms the algorithms become unusable due to severe throughput loss. I reserve this problem for future work.

Chapter 6

The Bandwidth Delay Product (BDP) Algorithm

6.1 Introduction

The problem commonly called *bufferbloat* is the result of poorly applied queuing theory. Setting the queue size too large causes excessive delay. Setting the queue size too small causes loss of throughput. Queuing theory gives us the bandwidth delay product equation, but, using this equation in the Internet has proven to be an extraordinarily difficult problem. Active Queue Management (AQM) algorithms have been developed to address this problem. The weakness with these AQM algorithms is that they are not truly parameterless, but, require some tuning to specific operating conditions. Our contribution consisting of our Bandwidth Delay Protocol AQM (BDP) is truly parameterless and able to adapt to any operating conditions without user intervention.

Packet switched networks such as the Internet require queuing because packet arrival time is non-deterministic. There is no guarantee that an arriving packet will be served immediately. Instead it will wait in a queue until forwarding service is available. Properly sizing the queue is a science that is deceptively complex. We have the

bandwidth delay product equation which says that queue size should be equal to the product of throughput and delay, Villamazar et al. [86]. If I could reliably predict the throughput and delay values in advance then I could perfectly size the queue for each TCP flow. However, in practice, these values have proven to be extraordinarily difficult to predict in advance.

Packet scheduling is a sister science orthogonal to queue sizing. Packet scheduling algorithms decide which packet to send next while queue management algorithms decide how long the queues should be [8, 76, 24, 52, 7, 78]. Packet scheduling algorithms do not solve the queue management problem and queue management algorithms do not solve the packet scheduling problem. AQM algorithms and packet scheduling should always work in tandem. In fact, in many cases the two algorithms are built into the same kernel module.[1] The important thing about packet scheduling algorithms (from an AQM perspective) is their ability to separate queues into individual flows without which AQM would be unmanageable in practice.

Loss based TCP congestion control algorithms come in many flavors with varying levels of aggression [17, 65, 61, 10, 56, 25]. Algorithms commonly deployed and active on the Internet today include Cubic (Linux), Compound (Windows XP), and NewReno (mac, Windows Vista+), [19, 68, 80]. Loss based TCP protocols seek to fill the slowest queue as much as possible and tend to increase buffering. The ubiquitously deployed TCP Cubic and NewReno are loss based. TCP Compound is a hybrid of loss and delay based TCP. These three protocols are the most popular variants of TCP. However, the combination of using loss based transport along with large unmanaged queues has lead to a condition popularly called bufferbloat. The problem has grown worse as the range of throughput and RTTs serviced by a queue grows more diverse.

Delay based TCP protocols are sensitive to RTT and back off in the face of increas-

[1] http://man7.org/linux/man-pages/man8/tc-fq_codel.8.html

ing delay. The problem with this approach is that loss based TCP protocols do not back off causing delay based protocols to lose throughput when in competition with a loss based protocol such as Cubic or NewReno. Unfortunately on the Internet today (except in certain controlled conditions) there is always a loss based competitor. Examples of delay based TCP are Vegas, CAIA CDG, TCP-Nice and Fast, [85, 36, 82, 9]. Scavenger protocols turn this drawback into a feature scavenging throughput when there are no other competitors and backing off when others are using the throughput. LEDBAT is an example of a scavenging protocol in use with Bit Torrent[2] [75]. Many other TCP variants have been created over the years including equation based TCP [33], Multipath TCP [34], split TCP [41], and Network Coding TCP (NCTP) [79]. However, no TCP variant exists that can properly manage queue size on the today's Internet.

Explicit Congestion Notification (ECN) is a specification that enhances the performance of TCP by marking packets rather than dropping them. Using ECN routers mark packets rather than drop them. The TCP sender reacts to ECN marked packets as if they were dropped packets [32, 72]. The state of deployment and activation of ECN in the Internet is poor but improving rapidly, about 40 percent at the core as of this writing [4, 51]. The problem is that many middleboxes (especially proxies) do not duplicate the ECN option properly.

The options field in the header is the mechanism designed to accommodate extensions to the TCP protocol. Middleboxes that do not properly duplicate the options field when copying the TCP header defeat the ECN extension to the TCP protocol [37]. However, if the sender requests ECN and the receiver reply's ECN okay then the ECN standard is implemented throughout the Internet path. Using ECN it is possible for a flow to have zero dropped packets. If the ECN reply is okay then ECN should be used.

[2]http://www.bittorrent.com/

An AQM algorithm must be able to use ECN if requested or drop packets if it is not.

The queue sizing problem has been around for a long time. There has been a lot of studies investigating the problem both in the core (Internet backbone) and at the edge (near the consumer), [39, 6, 87, 88, 47, 16, 62]. The key difference between the core and the edge is that in the core it is expected that there will be a large number of long lived TCP flows therefore the queue size will be on the order of $O\left(\frac{Capacity}{\sqrt{numberofflows}}\right)$, [3]. In contrast, on the edge I expect to have a more sparsely distributed traffic pattern and the queue size will be on the order of $Capacity * Delay$, [86].

AQM algorithms are designed to manage queue size. The IETF currently recommends that an AQM algorithm should be implemented in network devices in compliment with a DiffServ scheduler.[3] Examples of AQM algorithms include CoDel, PIE, ARED and many others, [22, 21, 20, 23, 48, 64, 66, 69, 93]. Two problems exist with the current generation of AQM algorithms. They control only a single queue at the IP layer while buffering may be displaced to another queue either vertically up and down the stack or horizontally across the network path to another device. The current generation of AQM algorithms cannot manage a truly large range of RTTs (from 10 ms to 1000 ms).

Over the last few decades AQM algorithms have evolved to adapt to these problems. Three algorithms represent the state of the art in queue sizing today. They are Adaptive Random Early Discard (ARED), Constant Delay (CoDel) and Proportional Integral controller Enhanced (PIE), [22, 64, 69]. ARED requires advance knowledge of both the throughput and delay for tuning. Tuning this algorithm is more of an art than a science and the consequences of improperly tuning the ARED algorithm can be quite severe: either excessive queuing delay or loss of throughput. This is the primary reason why despite having been deployed in routers for decades ARED and its cousins have

[3]http://tools.ietf.org/html/draft-ietf-aqm-recommendation-03

not seen widespread activation.

CoDel and PIE are more recent works that attempt to address the parameter problems encountered by ARED. Though these algorithms were originally advertised as *parameterless*, in practice they actually target a predetermined delay. This is a large improvement over ARED, however, they are still not *parameterless*. Our contribution to the field of queue sizing in this chapter is our Bandwidth Delay Protocol (BDP) AQM. Our BDP AQM requires no parameters self tuning to any bandwidth and delay. I built a prototype of our BDP algorithm as well as a testbed (including live Internet links) to demonstrate the flexibility and *parameterless* function of our BDP AQM algorithm in comparison to ARED, CoDel and PIE.

Calculating throughput in advance (as required for ARED) is a difficult prospect. The ISP gives us a "committed" rate in their Service Level Agreement (SLA). However, I know that the actual throughput can (and often does) fall below the "committed" rate during peak traffic times (from 7:00pm to 11:00pm).[4] In addition, the actual rate can exceed the "committed" rate during non-peak traffic hours because of Hierarchical Token Bucket (HTB) borrowing or PowerBoost technologies, [5].[5] This difficulty combined with the potentially extreme consequences of getting the queue size wrong (loss of throughput or excessive delay) have make it unlikely that any algorithm requiring accurate prediction of the throughput parameter unlikely to ever see widespread activation.

CoDel and PIE eliminate the need to predict throughput in advance by calculating queue time dynamically. Unfortunately due to the difficulty in predicting the actual RTT of a flow these algorithms target queuing delay to a static parameter. This can cause loss of throughput when the natural RTT (without bufferbloat) differs from the

[4]http://data.fcc.gov/download/measuring-broadband-america/2014/2014-Fixed-Measuring-Broadband-America-Report.pdf

[5]http://linux.die.net/man/8/tc-htb

targeted parameter. Our BDP AQM algorithm uses a novel active sensing mechanism to calculate the natural RTT of a flow (without the bufferbloat) and then uses a novel corrective algorithm in order to correct the queue size to match Villamazar's equation. Our BDP AQM algorithm correctly matches the queue size to any flow with any throughput and any RTT without the need for parameters.

6.2 Our Bandwidth Delay Protocol (BDP) Algorithm

I designed our BDP AQM algorithm to solve the problem of creating a truly parameterless AQM algorithm that can handle any RTT and any throughput. In order to do this I dynamically sense two values; the measured RTT and the bloat RTT. Using these two values I am able to calculate when AQM marking/dropping action should take place for any RTT. Also using these two values I am able to calculate an appropriate marking/dropping rate for any RTT. In this chapter I use cable modem technology for an example, however, our algorithm is flexible enough to handle most if not all network architectures.

Our BDP algorithm is shown in Figure 6.1 configured at the CPE modem in the upload direction. The download direction has a similar configuration. I note that it is also possible to configure our BDP algorithm at the ISP router, but, normally AQM algorithms are deployed at the modem. Packets arriving at the CMTS/DSLAM router encounter Differentiated Services Code Points, DSCP separating the Expidited Forwarding (EF) marked (VOIP low latency low throughput) from the Assured Forwarding (AF) or best effort queue. The EF queue needs no further processing and is immediately forwarded. The best effort queue is further classified by our BDP flow

classifier.

Our BDP flow classifier separates packets into flows by hashing on a triplet consisting of IP address src/dst and IP Flow ID. This hashing separates each flow into its own queue. At first glance this may seem an undue burden on the router resources, however, the hashing algorithms used are very efficient and produce little extra load on the CPU. In addition, the memory requirements are not increased because each packet must be stored for forwarding anyways regardless of classification. In any case most modern routers are already using some sort of classification algorithm.

The next step is to collect the RTT values. Beginning with the measured RTT value ($RTT_{Measured}$) I take a passive RTT measurement for each flow. In the past there was some concern about the CPU load incurred when doing RTT calculations. However, this is no longer a problem since the calculation has been reduced to only a few operations per packet and CPUs have become more powerful. The calculation from RFC 6289 is as follows:[6]

$$RTT_{var} = (1 - beta) * RTT_{var} + beta * |SRTT - R| \tag{6.1}$$

$$SRTT = (1 - alpha) * RTT_{var} + alpha * R \tag{6.2}$$

Where R is the RTT sample, RTT_{var} is the variance in RTT measurements, $SRTT$ is the smoothed RTT estimate. The recommended values for the constants are $alpha = \frac{1}{8}$ and $beta = \frac{1}{4}$. This CPU load is well within the capabilities of modern router CPUs.

The bloat RTT value (RTT_{Bloat}) can be collected in a number of ways: (1) ICMP packets, (2) SNMP packets, (3) Self addressed dataless IP packets. In this work I use

[6]https://tools.ietf.org/html/rfc6298

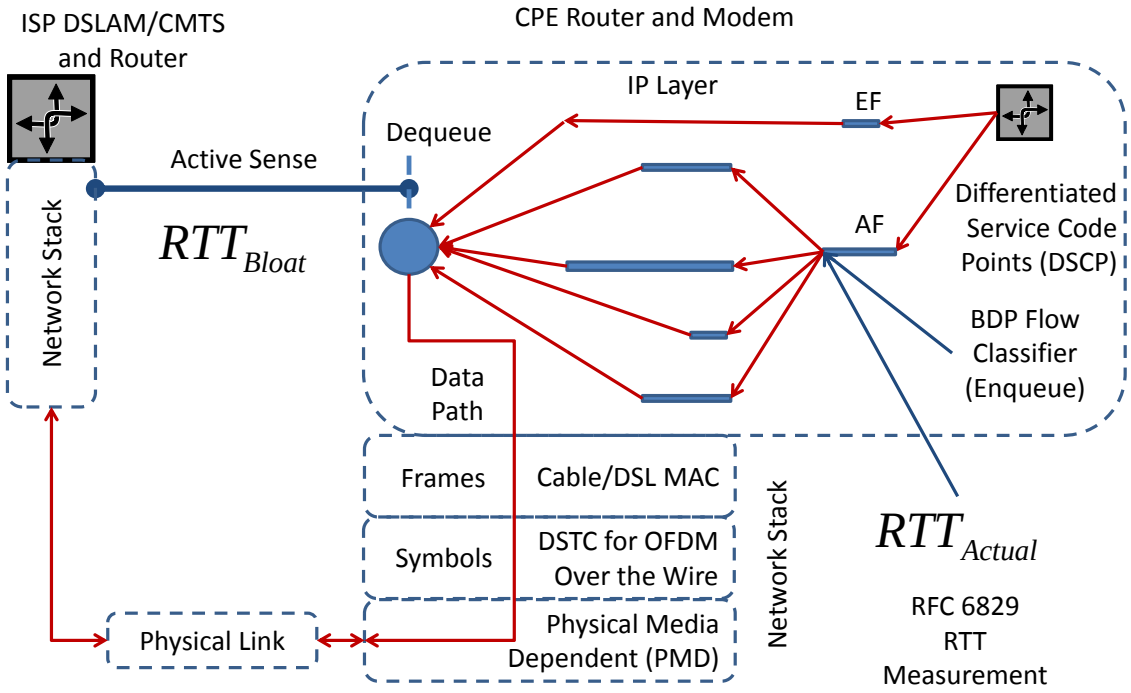


Figure 6.1: BDP AQM Algorithm (Download Direction)

(3), [35].[7] A self-addressed IP packet with a timestamp for payload is sent for every four packets on each flow. Each sensory packet is 28 bytes (20 bytes for the IP header + 8 bytes for the timestamp). This translates to less than 0.5% overhead with a 1500 byte MTU and less for larger MTU sizes. These packets are forwarded using an *iptables* rule.[8] The RTT is then calculated using the smoothing algorithm from RFC 6829.[9] As shown in Equation 6.1 and Equation 6.2 this requires only a few operations and well within the capabilities of modern CPUs.

Having collected both of the values that I needed ($RTT_{measured}$ and RTT_{Bloat}) I

[7]Under submission to IFIP Networking 2015
[8]http://linux.die.net/man/8/iptables
[9]https://tools.ietf.org/html/rfc6298

calculate the drop threshold.

$$RTT_{Bloat} > \min\left(RTT_{Measured}/10, 25\right) \tag{6.3}$$

If Equation 6.3 is true then our BDP AQM algorithm enters drop mode. This alows the queue to build up to about 10% of the flows RTT. Queues need to have some amount of variability in order to absorb short term bursts. Allowing the queue to fill to a percentage of the total RTT is better than targeting a fixed queuing delay because it adapts the amount of allowable delay to the RTT of the flow. Larger RTT flows require more queue in order to absorb short term bursts than smaller RTT flows. I have experimented with many different values and 10% represents a good tradeoff between responsiveness and cutting the queue too short. The 25 ms minimum queuing delay is designed to prevent the queue target from becoming too small and causing loss of throughput.

Next our BDP AQM algorithm calculates the interval of time that should elapse between marked/dropped packets. This calculation is very important because our BDP AQM algorithm is designed to operate on very small RTTs as well as very large RTTs. A linear drop ratio of $1/sqrtn$ where n is the number of dropped packets (used in CoDel) works well with small RTTs, but, is much too aggressive for large RTTs. A few observations provided insight into in the design of our interval algorithm. (1) The interval must scale inversely with the RTT (become less aggressive as the RTT grows larger. (2) If the bufferbloat is large in comparison to the measured RTT then the interval should become more aggressive. (3) There is no reason to ever mark/drop more than 1 packet per RTT (TCP will only respond to 1 drop per RTT anyways).

$$DROP_{Ratio} = RTT_{Measured}/RTT_{Bloat} \tag{6.4}$$

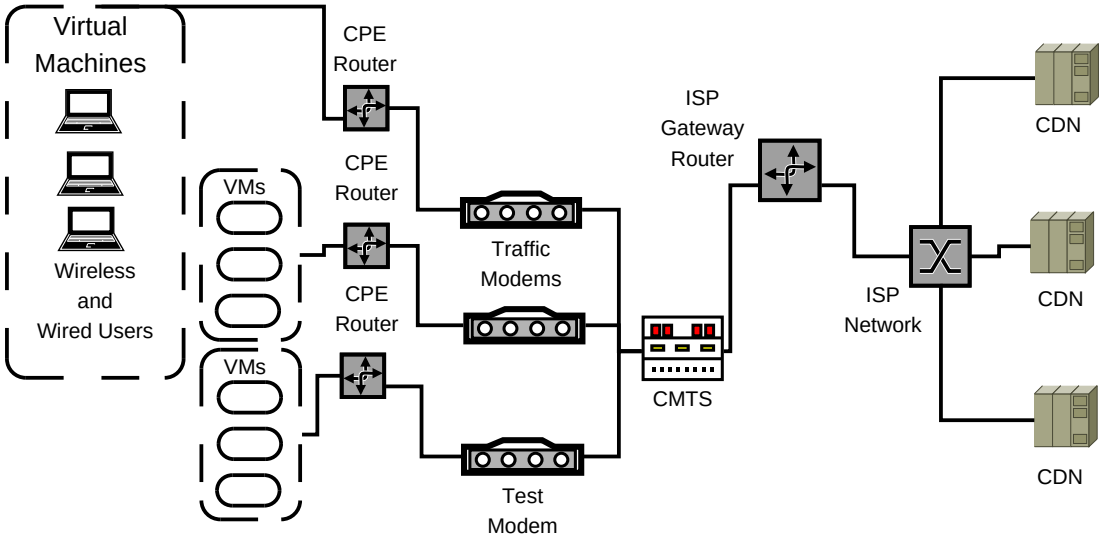


Figure 6.2: BDP Hardware Emulation Testbed

$$NEXT_{Drop} = RTT_{Measured} * DROP_{Ratio} \tag{6.5}$$

My solution for our BDP AQM algorithm was to use a calculated variable that I called $DROP_{Ratio}$. A large $DROP_{Ratio}$ is less aggressive and a small $DROP_{Ratio}$ is more aggressive. Equation 6.4 ensures that the $DROP_{Ratio}$ becomes less aggressive as the RTT increases and that it becomes more aggressive as the amount of bufferbloat increases. Equation 6.5 ensures that the marking/dropping interval is never less than 1 RTT.

6.3 BDP Testbed

The goal of our evaluation was to demonstrate the parameterless flexibility of our BDP AQM algorithm by comparing and contrasting it to the best AQM algorithms available today. I hope to show that the parameterless flexibility of our BDP AQM algorithm allows it to operate in all conditions without user intervention even when

other algorithms require re-tuning. In order to facilitate our evaluation goals I constructed the testbed shown in Figure 6.2. The testbed was constructed on the Flexnet's Emulab platform for academic research [91]. Flexnet's Emulab is a cloud platform for networking research constructed of 1/10 Gigabit as well as multicore PC's and virtual LANs.

In Figure 6.2 the (download) data flows from the Content Delivery Networks CDNs (emulated by pcs) to the ISP network and the ISP gateway router. The ISP gateway router forwards the data streams to the CMTS node which channelizes the data over the cable broadband link to each modem. The modem delivers the data streams to CPE routers which forward the data streams to the virtual machines (constructed from three pcs). Upload data flows in the opposite direction.

The modems, CPE routers, CMTS, and ISP gateway were constructed from pcs running the linux 3.17 kernel. The modem nodes were equipped with an array of AQM algorithms including ARED, CoDel, PIE and our BDP algorithm. Each modem has an Hierarchical Token Bucket (HTB) rate limiter as is common with Cisco and other manufacturers equipment.[10] Each experiment run time was 120 seconds with 5 experimental runs compiled into a CDF. The next step was to compile a set of graphs from the thousands of experiments that I conducted to compare and contrast our BDP AQM algorithm with ARED, CoDel and PIE.

In our BDP experiments, traffic is generated from CDNs to virtual user machines running the Linux 3.2 kernel. The traffic across each modem was generated according to the AQM Evaluation Guidlines IETF draft; five repeating TCP transfers of 5MB each, one continuous TCP transfer and four HTTP web traffic (repeated downloads of 700kB).[11] This traffic mix was designed to investigate the effects of bufferbloat with a

[10]http://linux.die.net/man/8/tc-htb
[11]http://tools.ietf.org/html/draft-kuhn-aqm-eval-guidelines-00#section-3.2.4

mixture of long term and short flows in combination.

6.4 Evaluation

In this evaluation I sought to demonstrate the flexibility of our parameterless BDP AQM algorithm. I did this by comparing our BDP AQM algorithm to the best AQM algorithms available today: ARED, CoDel and PIE. I want to demonstrate that our BDP AQM algorithm can auto-tune itself to adapt to any operating conditions even those that would require re-tuning of the other algorithms.

I have run thousands of experiments with a wide variety of parameter settings w.r.t. RTT and Throughput (RTTs from 10 ms to 1000 ms) and throughputs from 2 Mbps to 50 Mbps). Of the thousands of graphs generated I present a subset of the CDFs generated. I present CDFs from a series of experiments with the modems from the testbed in Figure 6.2 configured for 16 Mbps committed rate and 32 Mbps peak rate. The access link was configured for a rate of up to 48 Mbps because that is enough to provide full throughput to all three modems at the same time.

I present graphs with RTTs from 100 ms to 1000 ms. I chose not to present graphs with RTTs less than 100 ms because 100 ms and less RTT are very favorable conditions. All four algorithms performed similarly well in these favorable RTT settings achieving very nearly full throughput and minimum RTT. I also chose not to present graphs at RTTs above 1000 ms because these RTT conditions are so terrible that the flows become severely TCP limited and do not need AQM protection.

I tuned the ARED algorithm for 100 ms with 16 Mbps throughput (the committed rate of the modem) as recommended in the documentation. This corresponds to a queue size 16 $Mbps * 100\ ms = 1.6\ Mbits = 200,000\ Bytes$. This corresponds to the following parameters $minimum = 70000$, $maximum = 140000$ and $limit = 210000$.

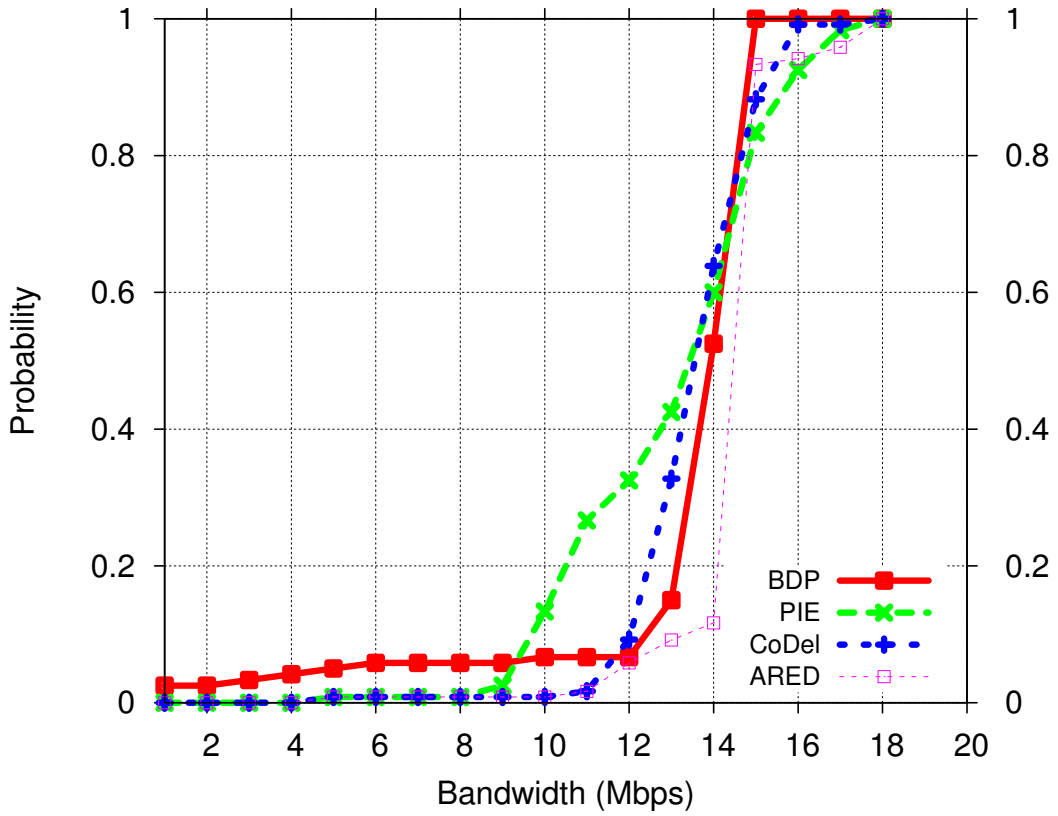


Figure 6.3: AQM Throughput at 100 ms RTT

I ran CoDel with its default parameters $target = 10\ ms$ and $interval = 100\ ms$ and PIE with its default parameters of $target = 20\ ms$ and $tupdate = 30\ ms$. Our BDP algorithm is parameterless and has no default settings.

All of the experiments include the slow start phase of Cubic TCP and run for 120 seconds allowing time for the TCP algorithm to settle. In Figure 6.3 I present the throughput curves for ARED, CoDel, PIE and BDP. All four algorithms performed well in this experiment because 100 ms is well within their operating ranges. It was no surprise that ARED performed the best since it is exactly what ARED was tuned for. Our BDP algorithm was the next best contender with CoDel coming in next and PIE following a little behind. The performance differences between the algorithms in these vary favorable conditions were detectable but insignificant from the perspective

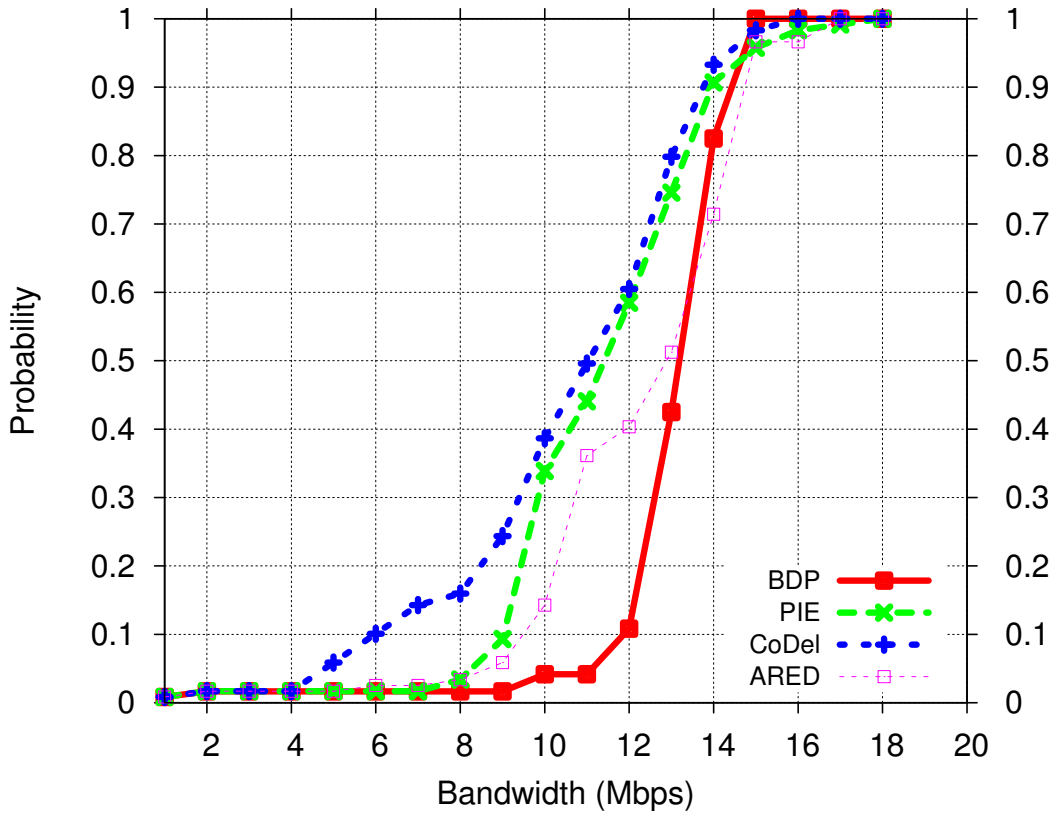


Figure 6.4: AQM Throughput at 250 ms RTT

of user performance. All four did very well.

At 16 Mbps a TCP flow can be expected to achieve about 12 Mbps to 15.5 Mbps because of overhead (packet headers) and TCP contention with the other flows. The most interesting thing to note from the CDF in Figure 6.3 is the straightness of the lines for ARED and our BDP AQM algorithm. The curves for CoDel and PIE were not nearly so straight. This effect is caused the AQM algorithm correcting (slightly) too aggressively when the queue built up. This drains the queue quickly but causes a slight loss of throughput. Once the queue is drained the flow refilled the queue very aggressively allowing the flow to temporarily exceed its committed rate allowing PIE, CoDel and ARED to achieve 16 Mbps to 18 Mbps temporarily. Our BDP AQM algorithm did not exhibit this problem.

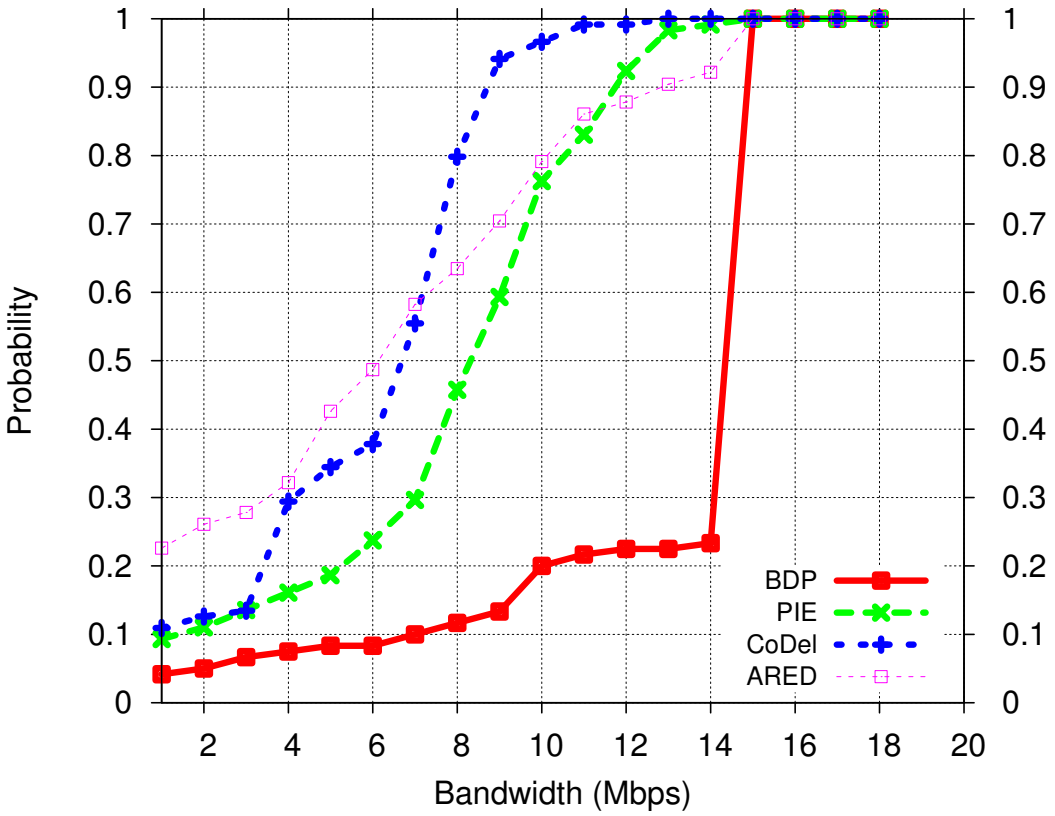


Figure 6.5: AQM Throughput at 500 ms RTT

In Figure 6.4 I show the throughput curves for the algorithms at 250 ms RTT making the environment a little more challenging. The CDF shows that other than slow start our BDP AQM algorithm was virtually unaffected. The curve is nice and straight, does not exceed the maximum expected throughput (about 15.5 Mbps) and does not go below the minimum expected throughput (about 12 Mbps). The small portion of the line (less than 10% of the packets) that is below 12 Mbps is due to slow start. The ARED algorithm came in the next best achieving the minimum expected (or more) throughput of 12 Mbps for 60% of the packets. This is because the 16 Mbps at 250 ms RTT was not too far from AREDs tuning of 16 Mbps at 100 ms. However, a trend of losing throughput due to overly aggressive AQM correction is beginning to show.

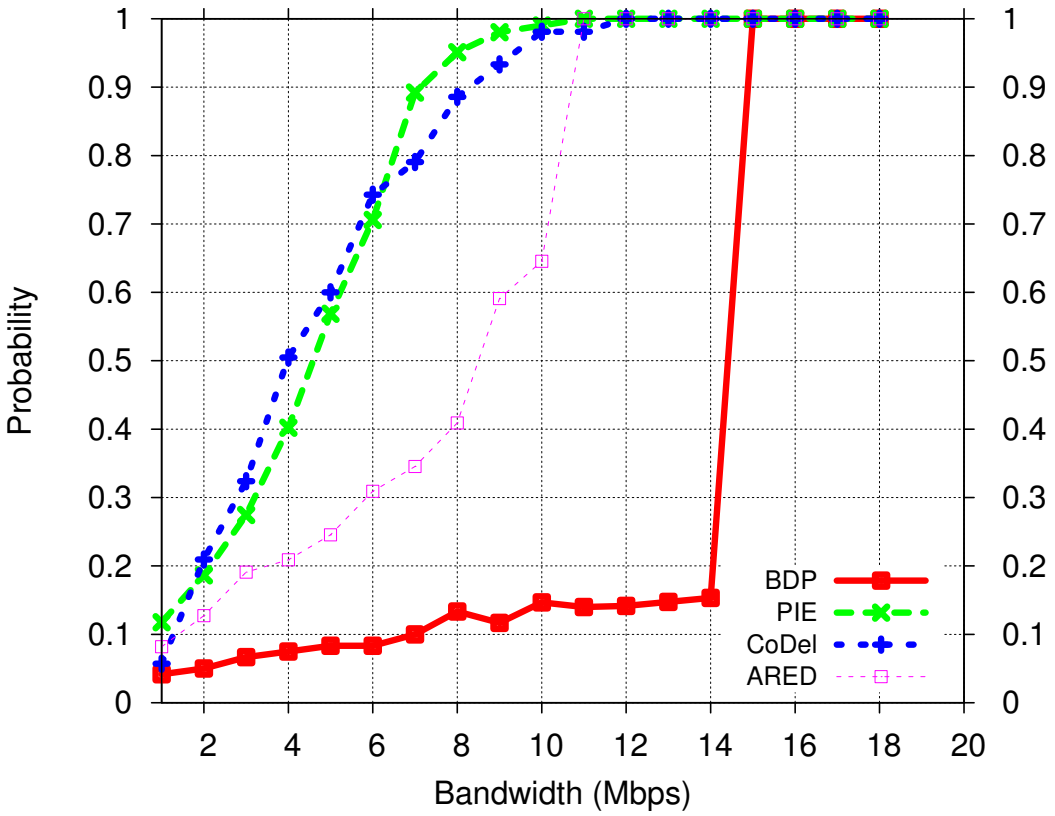


Figure 6.6: AQM Throughput at 750 ms RTT

The PIE and CoDel algorithms did not fare nearly as well. Both of these algorithms are losing throughput due to aggressive AQM correction. Neither PIE nor CoDel achieved the expected minimum 12 Mbps throughput for 60% of the packets. This throughput performance for PIE and CoDel was mediocre and the performance of ARED was slightly better. Clearly all three algorithms are beginning to lose throughput and only our BDP AQM performed perfectly. This performance by BDP was because BDP was able to auto tune its parameters to adapt to the changing RTT conditions.

In Figure 6.5 I present the throughput CDFs for the algorithms at 500 ms RTT. This is the maximum of the operating range specified by the documentation for CoDel and PIE. ARED does not have an operating range since it is meant to be tuned to

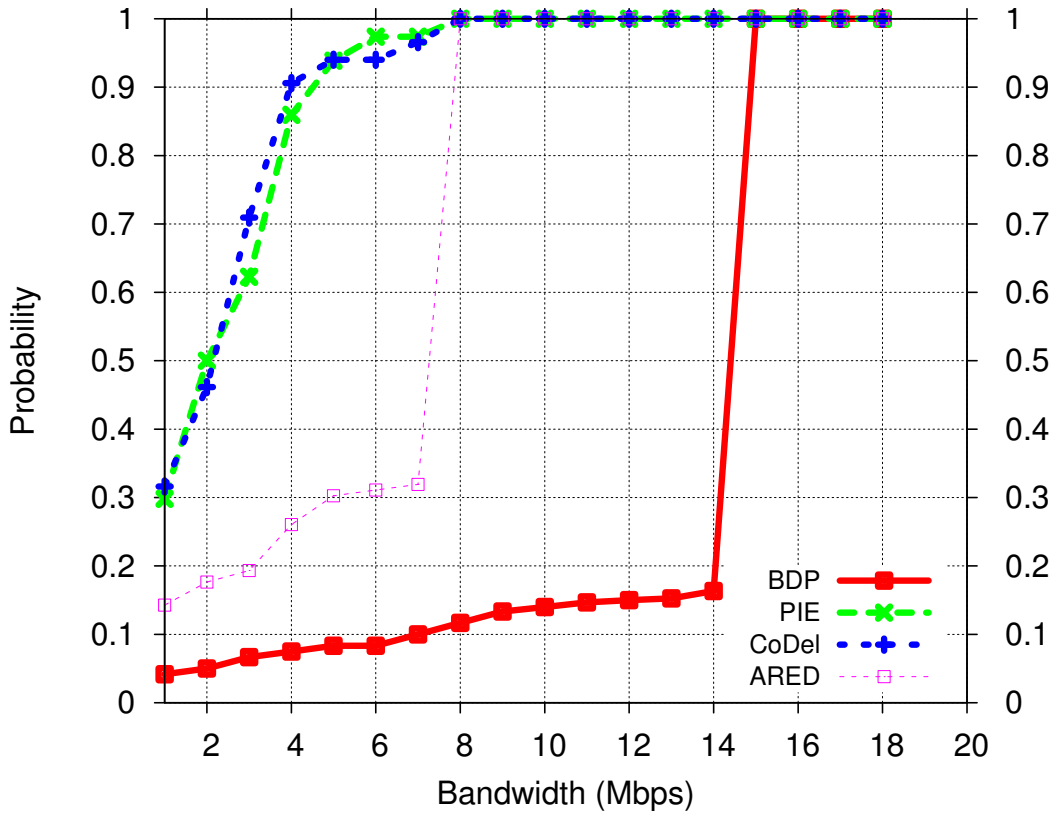


Figure 6.7: AQM Throughput at 1000 ms RTT

specific operating conditions and of course our BDP AQM algorithm is parameterless and auto-tunes itself to any operating conditions. Our BDP AQM algorithm was able to auto-tune itself to the challenging RTT conditions. After slow start (about 22% of the packets) our BDP AQM algorithm operated between 14 Mbps and 15.5 Mbps well within the expected minimum (12 Mbps) and maximum (15.5 Mbps). Once again our BDP AQM algorithm turned in a near perfect performance demonstrating its parameterless flexibility.

ARED, CoDel and PIE did not fare well in this more challenging RTT scenario. The ARED algorithm only managed about 6 Mbps for 50% of the packets clearly ARED is too far out of its tuning and is much too aggressive with its AQM correction. This performance by ARED was expected because AREDs operating range is much

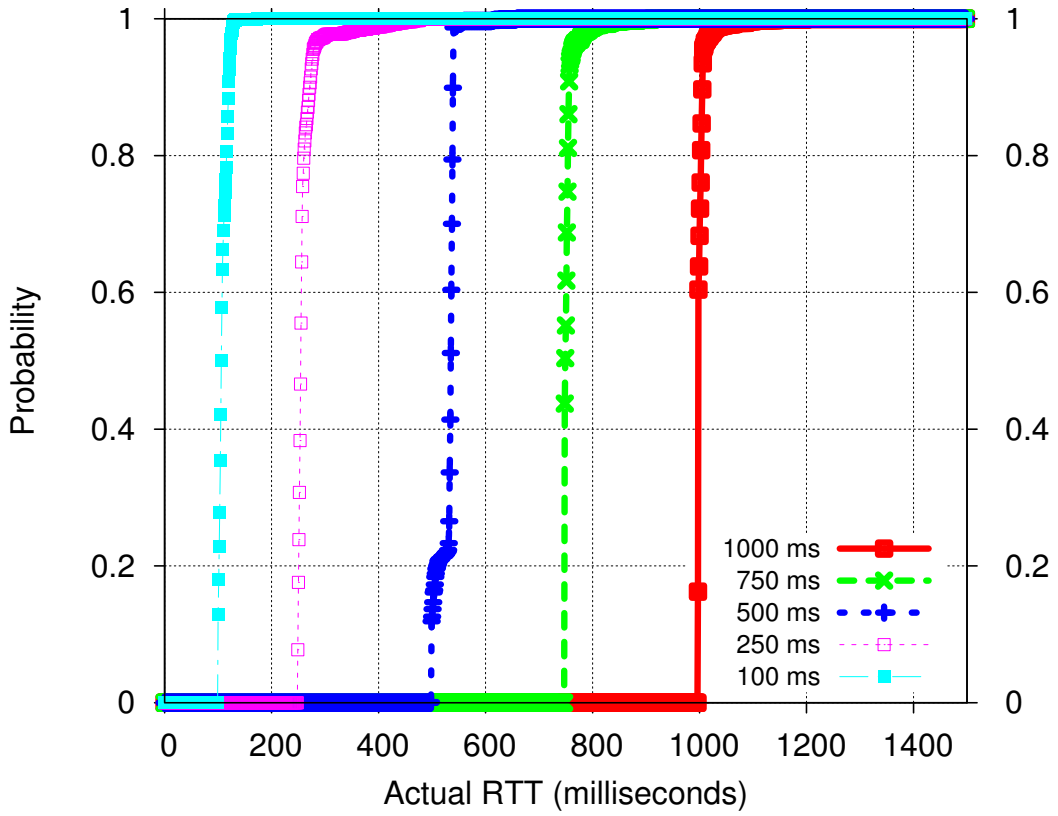


Figure 6.8: BDP AQM RTT CDF

narrower (though not clearly defined) than CoDel or PIE. However, CoDel and PIE did not perform much better with CoDel managing about 7 Mbps and PIE about 8 Mbps for 50% of the packets. The loss of throughput experienced by ARED, CoDel and PIE was severe (nearly half) and noticeable. Users operating these algorithms in the 500 ms RTT condition would clearly notice a degradation in throughput performance. Although the degradation in throughput was expected in the case of ARED, it is surprising in the case of CoDel and PIE because both of these algorithms are supposed to be able to operate at this RTT.

In Figure 6.6 I present the throughput CDFs for the algorithms at 750 ms RTT. At 750 ms RTT TCP is RTT limited and the maximum throughput is 11 Mbps with a minimum of about 8 Mbps. The CDF shows that once again our BDP AQM algorithm

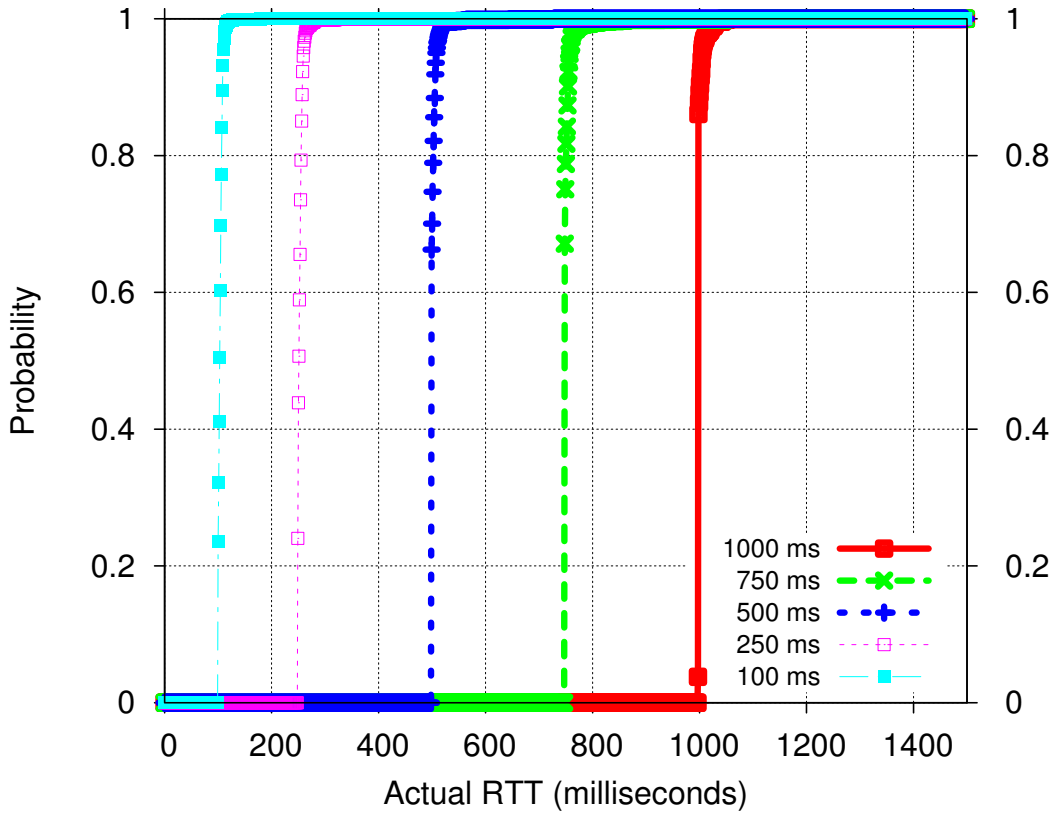


Figure 6.9: CoDel AQM RTT CDF

goes through slow start (about 22% of the packets) and then reaches a stable operating throughput between 8 Mbps and 11 Mbps. Our BDP AQM algorithm adapted to these extremely challenging operating conditions reaching the maximum operating throughput.

The most interesting thing about this CDF is the curve for ARED. The ARED algorithm actually fared better at 750 ms RTT than at 500 ms. This was because the TCP RTT limitation reduced the throughput ARED was expecting a 16 Mbps flow and it encountered a 10 Mbps flow. The reduced throughput brought ARED closer in tune. The CoDEL and PIE algorithms did not fare nearly as well with 50% of the packets at about 4 Mbps or less. These algorithms were clearly not expecting to encounter this large of an RTT and are too aggressively applying AQM corrective action.

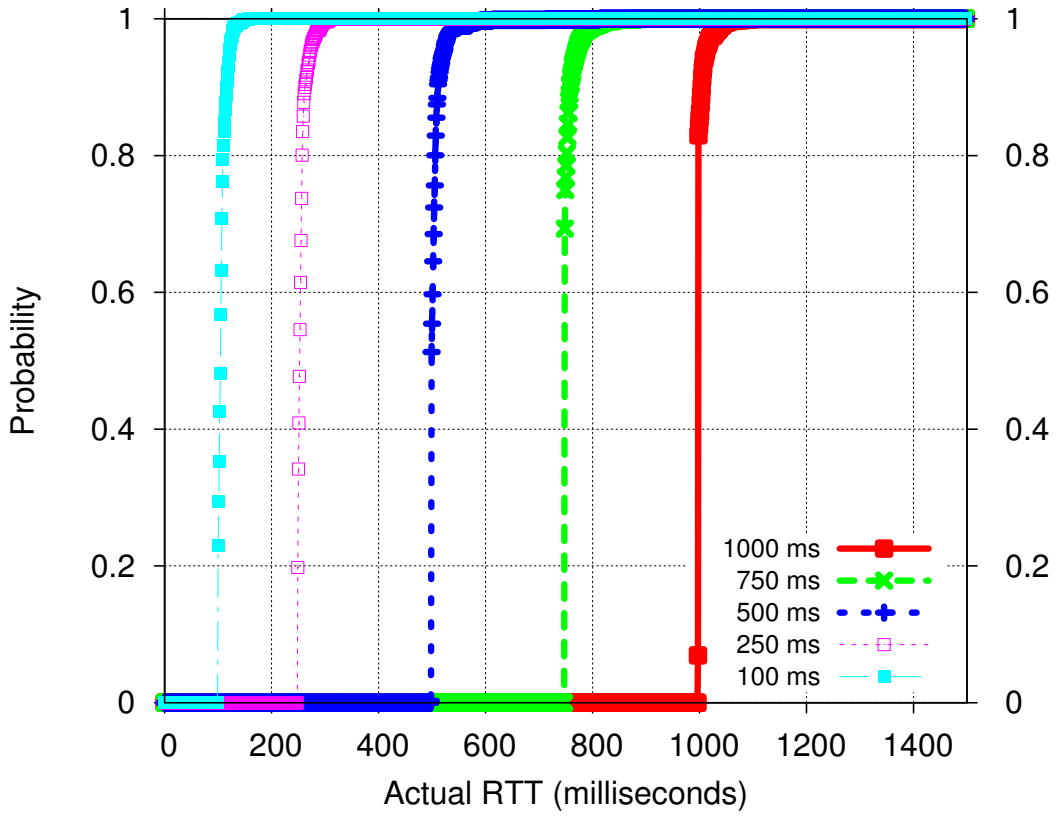


Figure 6.10: PIE AQM RTT CDF

I present our last throughput CDF in Figure 6.7 at 1000 ms RTT. This is about as large of an RTT as can reasonably expected during typical Internet usage (although multi-hop satellite links can have larger RTTs). TCP is further RTT limited at this RTT to between about 7 or 8 Mbps. Our BDP algorithm auto-tuned itself to adapt to the challenging conditions achieving the maximum possible throughput in these extremely challenging RTT conditions. The ARED algorithm also did surprisingly well having been helped out by the further reduction in throughput. CoDel and PIE both suffered from extreme loss of throughput with 90% of the packets achieving 4 Mbps or less because of too aggressive AQM correction.

The next step is to examine the RTT characteristics for each of the four algorithms. I present the RTT CDFs for each of the four algorithms separately for the sake of clarity.

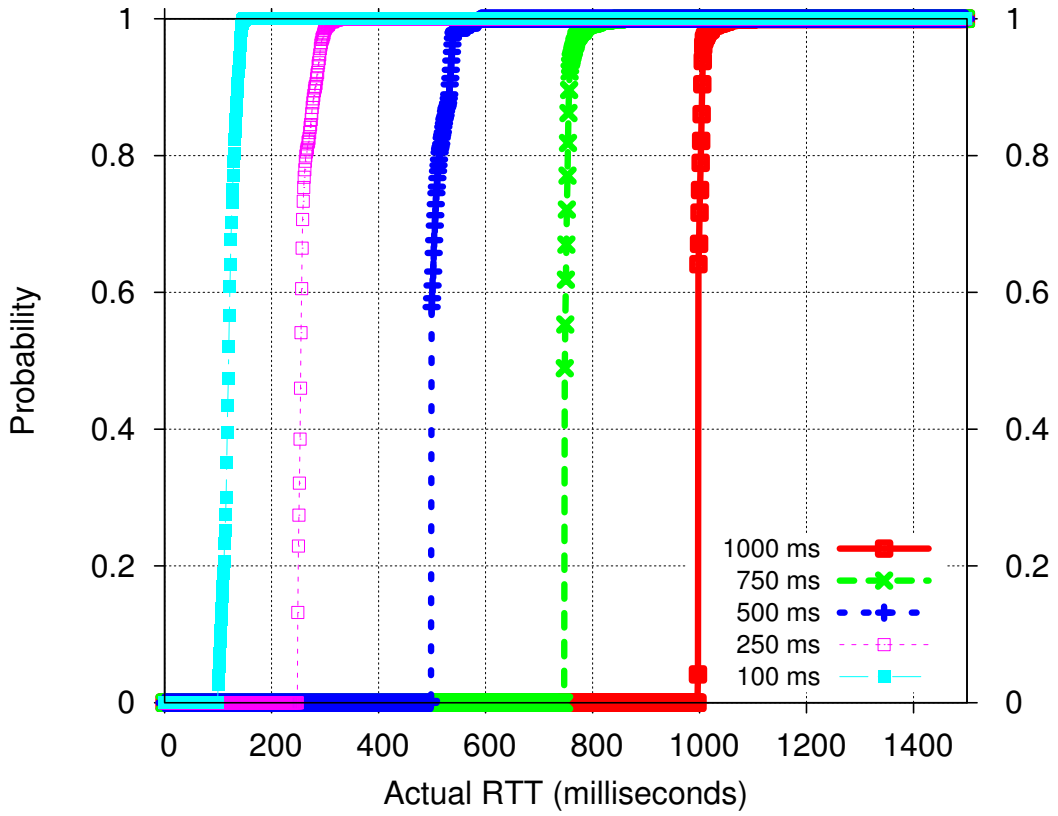


Figure 6.11: ARED AQM RTT CDF

I present BDP in Figure 6.8, CoDel in Figure 6.9, PIE in Figure 6.10 and ARED in Figure 6.11. The RTT characteristics for each algorithm are shown at 100 ms, 250 ms, 500 ms, 750 ms and 1000 ms. There are minor differences between the RTT characteristics of each algorithm. However, these differences are insignificant. This is the point of presenting all four graphs to demonstrate that each algorithm performs excellently in terms of RTT.

6.5 Conclusions and Future Work

In this work I have presented our parameterless auto-tuning BDP AQM algorithm. Our BDP AQM algorithm is able to adapt itself to any throughput and any RTT

that it might encounter by calculating the RTT of a flow and separating it into two components, the bloat RTT and the natural RTT. Using this information our BDP AQM algorithm is able to determine whether a flow is actually bloated or if it just has a large RTT and to calculate a marking/dropping interval that is adaptive to the RTT of a flow.

I compared our BDP AQM algorithm to the three best AQM algorithms available today; CoDel, PIE and ARED. All four of the algorithms performed excellently in terms of RTT across a large range of RTTs up to 1000 ms. In terms of throughput the ARED algorithm did surprisingly well, but, has the drawback of requiring specialized tuning for specific operating conditions. CoDel and PIE require less tuning and are able to operate over a larger range of operating conditions than ARED, however, as our experiments have shown they suffer from significant loss of throughput at RTTs above 250 ms that would require significant tuning to fix.

Our BDP AQM algorithm is truly parameterless and needs no tuning regardless of operating conditions. I have conducted thousands of experiments with many throughputs (from 2 Mbps to 50 Mbps) and many RTTs (from 10 ms to 1000 ms). Our experiments have shown that our BDP AQM algorithm gets excellent results in terms of throughput and RTT throughout any range of operating conditions. Of this extremely large set of graphs produced by our experiments I chose to present a set of throughput and RTT CDFs at 16 Mbps and RTTs from (100 ms to 1000 ms).

With the exception of ARED all of these AQM algorithms are throughput agnostic and other throughput settings will produce similar results. I chose 16 Mbps because this throughput is representative of a typical consumer access link in America today according to the FCC study Measuring Broadband in America 2014. I did not choose to present CDF's below 100 ms RTT because all of the algorithms perform well in these favorable conditions. I also chose not to present CDFs above 1000 Mbps because

TCP is severely RTT limited under these challenging conditions.

I investigated the use of our BDP AQM algorithm implemented at the CPE modem because this is a typical configuration used in consumer access links. However, other configurations are possible. Our BDP AQM algorithm could be implemented at the ISP gateway router or in the user device in order to serve wireless broadband connections. In addition, AQM algorithms can have interesting interactions with scavenging protocols. I leave this investigation to future work.

Chapter 7

Conclusions

Internet services have become a a vital part of the lives of billions of people worldwide. Providing these services twenty four hours a day seven days a week had become indispensable to the quality of life for people across the planet. Ensuring that these services are maintained and that the needs of future services met requires that the Internet's underlying services remain healthy and in good working order. In order to maintain the health of the Internet I have identified two key metrics that indicate the health of the Internet. These metrics are bandwidth and delay. Bandwidth and delay correspond to how many Internet objects can be delivered over time. Regardless of how much of this resource is actually used these two metrics indicate how much Internet resource is available. The availability of Internet resources is important because the resource could be used to deliver life improving services.

I have identified two key ways in which Internet resources are wasted. This waste occurs when the amount of packet loss is so great that it overcomes the systems that have been designed and deployed to overcome loss are overwhelmed. Typically this amount of loss occurs over wireless links in rural areas where wireless connections are weak and there is much interference and in saturated metropolitan areas where collisions cause excessive packet loss. The second type of waste occurs when the network queue is improperly sized for the flows being serviced. Both of these types of waste

can cause extreme loss of bandwidth and excessive delay. In this dissertation I have developed new techniques that mitigate these both of these problems.

In order to address the loss of bandwidth caused by excessive packet loss in wireless systems that are operating in challenging environments either rural or metropolitan I have developed the Receiver Driven Rate Adaptation (RDRA) algorithm. RDRA is a parallel TCP system that increases robustness against packet loss. RDRA divides data over 8 TCP streams. Eight TCP streams are more robust against packet loss than a single TCP stream because losses during a single RTT only affect one of the streams. A comparison demonstrates this effect. A single stream TCP operating at 8 Mbps encountering packet loss will halve its congestion window reducing throughput to 4 Mbps. If that same 8 Mbps is divided over eight TCP streams operating at 1 Mbps when packet loss occurs it will reduce the throughput of 1 of the streams to 0.5 Mbps leaving the others operating at 1 Mbps for a total of 7.5 Mbps. Though reality does not always work this way and sometimes the loss of multiple packets hits multiple streams the distribution of load over multiple streams still increases robustness.

Parallel TCP creates an unfairness problem because of the multiple streams. A flow operating with 8 TCP streams will achieve 8 times the throughput as a flow operating with 1 TCP stream. RDRA addresses this problem by calculating the throughput expected of a single stream TCP and reducing the flow of each stream accordingly so that RDRA's throughput more closely matches the throughput of a single stream TCP. RDRA is both more robust against packet loss than a single stream TCP and more fair than a parallel TCP.

The second technique I developed to address the loss of bandwidth caused by excessive packet loss is called the Fast Wireless Protocol (FWP). The FWP system increases aggregation of frames in order to increase throughput and overrides the MAC layer retransmission in order to hide wireless packet losses from the transport layer above it.

The original goals of FWP had Single Input Single Output (SISO) systems in mind. However, it has turned out that Multiple Input Multiple Output (MIMO) and even Multi User-MIMO (MU-MIMO) systems have become overwhelmingly popular. In fact, increasing aggregation is bad for MU-MIMO. In light of this outcome I do not recommend using the increased aggregation called for in FWP. However overriding the MAC layer retransmission scheme is still a good idea. The FWP system hides frame loss from the transport layer by injecting temporary frames in place of frames lost during wireless transmission. This hides the loss from the transport system and prevents inappropriate backoff by the TCP. The frames are then retransmitted later from a session layer retransmission system.

The second problem addressed in this dissertation is the excessive delay and or loss of bandwidth caused by inappropriate queue sizing. The science of queue management is not new and two modern techniques already exist in order to reduce the excessive queuing part of the problem. However, two rather large areas of this discipline still remain unaddressed. These areas are the ability to address the queue sizing problem wherever it occurs in the access link, and the ability to calculate the right queue size regardless of the characteristics of the flow. I have designed two novel queue management techniques that address these problems.

The first technique that I designed addresses the problem of managing the queue size wherever it is found throughout the access link. Active Queue Management (AQM) techniques exist. but, they cannot control queues outside of the queue that they monitor in the IP layer. My algorithm is called Active Sense Queue Management (ASQM) and it uses an active sensing technique to discover the queue size throughout the access link rather than simply monitoring the size of the queue at the IP layer. This sensory mechanism allows ASQM to control queues throughout the access link even when the queuing occurs in layers below the IP. This is of particular importance

since the Federal Communications Commission (FCC) study over the years 2010-2014 has indicated that during peak hours ISPs often do not provide 100% of the rated bandwidth on their customer access links. When this happens queues within the lower layers build up causing excessive delay. ASQM monitors these queues and provides queue management at all times.

The second technique that I designed to address the queue sizing problem is called the Bandwidth Delay Protocol (BDP). The BDP algorithm is unique among AQM algorithms in that it separates individual flows into queues and calculates their bandwidth delay product producing a queue of the correct size for each flow. The BDP algorithm accomplishes this using a combination of active and passive sensory techniques to discover the specific queue size required for each flow. I took special care in crafting the active sensory technique reducing the overhead incurred to 0.5% or less. The BDP algorithm produces excellent bandwidth and delay characteristics for small delay flows up to 250 ms RTT (just as CoDel, PIE and ASQM) and continues to produce excellent results for larger delay flows from 250 ms to 1000 ms where the other algorithms lose bandwidth because they size the queues too small. The BDP algorithm can manage any flow at any bandwidth and any RTT.

These four algorithms address fundamental problems faced in the Internet today and problems that will be faced in the future. The RDRA and FWP algorithms address the problem of extreme packet loss causing inappropriate backoff and loss of bandwidth. This is a problem that will always be faced on the edge of the Internet where challenging wireless transmission characteristics cause a great deal of packet loss and in saturated metropolitan conditions where collisions also cause extreme packet loss. The ASQM and BDP algorithms address the queue sizing problem. The queue sizing problem is ubiquitous across the Internet since packet switching requires queuing and queuing requires queue sizing. The queue sizing problem can move about across

a link occurring at whichever queue is the slowest. The ASQM algorithm addresses this problem by managing the queue size across the entire link rather than at a single queue. The BDP algorithm addresses the problem of queue sizing for individual flows. Each flow has its own unique bandwidth delay product and requires a specific queue size tailored to its needs.

These four algorithms that I have developed address the fundamental problem of optimizing bandwidth and delay for all flows on the Internet. The first two algorithms RDRA and FWP ensure that wireless connections in the last mile faced with challenging conditions can achieve a closer share of their fair bandwidth. This addresses the important problem of empowering underserved flows allowing them more bandwidth. The second two algorithms ASQM and BDP address the queue sizing problem. This ensures that no flow on the Internet will be bandwidth limited because of its queue size regardless of its Round Trip Time delay characteristics and that no flow will either experience or cause excessive delay.

Between these four algorithms I have optimized the flow of data on the Internet such that each flow can obtain its fair share of the bandwidth resource at its minimum latency. By optimizing these two characteristics latency and delay I have provided for the efficient and effective delivery of Internet objects. The smooth and efficient delivery of Internet objects ensures the continued delivery of current and future life improving Internet services for billions of people around the world who rely on these services on a daily basis.

7.1 Future Work

It has been widely reported in the press that business disputes between providers have been resulting in congestion at interconnection points and network paths leading to reduced bandwidth and increased latency. I cannot solve this problem since

it is a business problem and not a scientific one and I can do nothing about the reduced bandwidth because it is intentionally causes the latency is a secondary effect and can be solved scientifically. Effectively what has happened is one provider has reduced the bandwidth for another provider and changed the bandwidth delay product. An appropriate change in the queue size will eliminate the increased latency. Using our ASQM techniques a provider experiencing this problem could send active sensory packets through the provider network that is slowing them down. This would discover the queue size necessary for the flow and the provider could adjust their queue size appropriately eliminating the excess latency.

The queue management techniques (ASQM and BDP) expect the core of the Internet to be faster than the edge of the Internet. This is normally the case and is generally accepted as fact. However, it is not a given that this will always be the case in the future. It could be that new access technologies outstrip the pace of improvement in core technologies causing the Internet topology to flip. This would result in the condition that the edge of the Internet is the faster part and that queuing occurs in the core. Our active sensory techniques could also be used in this case. The sender would need to participate sending active sensory packets to be returned by the faster edge router allowing the discovery of the correct queue size. This would allow the sender to size the queues appropriately minimizing delay and maximizing bandwidth.

Bibliography

[1] S. Akhshabi, A. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *MMSYS*, pages 157–168, 2011.

[2] E. Altman, D. Barman, B. Tuffin, and M. Vojnovc. Parallel TCP Sockets: Simple Model, Throughput and Validation. In *Infocom*, page 17. Microsoft Research, Sept. 2005.

[3] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. *SIGCOMM*, 34(4):281–292, Aug. 2004.

[4] S. Bauer, R. Beverly, and A. Berger. Measuring the State of ECN Readiness in Servers, Clients, and Routers. In *Internet Measurement Conference*, pages 171–180, 2011.

[5] S. Bauer, D. Clark, and W. Lehr. PowerBoost. In *HomeNets*, pages 7–12, 2011.

[6] N. Beheshti, Y. Ganjali, M. Ghobadi, N. Mckeown, and G. Salmon. Experimental Study of Router Buffer Sizing. In *IMC*, pages 197–210, 2008.

[7] J. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. *Networking*, 5(5):675–689, Oct. 1997.

[8] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. A Framework for Integrated Services Operation over Diffserv Networks. RFC 2998, Nov. 2000.

[9] L. Brakmo and L. Peterson. TCP Vegas: end to end congestion avoidance on a global Internet. *Communications*, 13(8):1465–1480, Oct. 1995.

[10] C. Caini and R. Firrincieli. TCP Hybla: a TCP enhancement for heterogeneous networks. *Satellite Communications and Networking*, 22(5):547–566, 2004.

[11] L. Chen. TFRC Modeling and Its Applications. Master's thesis, University of Hong Kong, China, 2003.

[12] M. Chen and A. Zakhor. Rate control for streaming video over wireless. In *INFOCOM*, pages 1181–1190, Mar. 2004.

[13] M. Chen and A. Zakhor. Flow Control Over Wireless Network and Application Layer Implementation. In *INFOCOM*, pages 1–12, Apr. 2006.

[14] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989.

[15] L. Cicco and S. Mascolo. An experimental investigation of the Akamai adaptive video streaming. In *HCI*, pages 447–464, 2010.

[16] A. Dhamdhere and C. Dovrolis. Open issues in router buffer sizing, 2006.

[17] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi. Proportional rate reduction for TCP. In *IMC*, pages 155–170, 2011.

[18] J. Erman, A. Gerber, K. Ramadrishnan, S. Sen, and O. Spatscheck. Over the top video: the gorilla in cellular networks. In *IMC*, IMC, pages 127–136, 2011.

[19] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. *SIGCOMM*, 26(3):5–21, July 1996.

[20] W. Feng, D. Kandlur, D. Saha, and K. Shin. A self-configuring RED gateway. In *INFOCOM*, volume 3, pages 1320–1328, Mar. 1999.

[21] W. Feng, K. Shin, D. Kandlur, and D. Saha. The BLUE active queue management algorithms. *Networking*, 10(4):513–528, Aug. 2002.

[22] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. *Preprint, available at http://www. icir. org/floyd/papers. html*, 2001.

[23] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Networking*, 1(4):397–413, Aug. 1993.

[24] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *Networking*, 3(4):365–386, Aug. 1995.

[25] C. Fu and S. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. *Communications*, 21(2):216–228, Feb. 2003.

[26] A. Gember, A. Anand, and A. Akella. A Comparative Study of Handheld and Non-handheld Traffic in Campus Wi-Fi Networks. In *Passive and Active Measurement*, volume 6579 of *Lecture Notes in Computer Science*, pages 173–183. 2011.

[27] A. Goel, C. Krasic, and J. Walpole. Low-latency adaptive streaming over tcp. *Transactions Multimedia Computing, Communications, and Applications (TOMCCAP)*, 4(3):1–20, Sept. 2008.

[28] h. Schulzrinne, G. Fokus, and S. Casner. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, Internet Engineering Task Force, Jan. 1996.

[29] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS*, 42(5):64–74, July 2008.

[30] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *Parallel and Distributed Processing Symposium*, pages 434–443, 2002.

[31] T. Hacker, B. Noble, and B. Athey. Improving throughput and maintaining fairness using parallel TCP. In *INFOCOM*, pages 2480–2489, Mar. 2004.

[32] T. Hamann and J. Walrand. A new fair window algorithm for ECN capable TCP (new-ECN). In *INFOCOM*, volume 3, pages 1528–1536, Mar. 2000.

[33] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. Technical Report 5348, Internet Engineering Task Force, Sept. 2003.

[34] S. Hassayoun, J. Iyengar, and D. Ros. Dynamic Window Coupling for multipath congestion control. In *ICNP*, pages 341–352, Oct. 2011.

[35] D. Havey and K. Almeroth. Active Sense Queue Management (ASQM). In *Under submission to IFIP Networking 2015*, 2015.

[36] D. Hayes and G. Armitage. Revisiting TCP Congestion Control Using Delay Gradients. In *Networking*, volume 6641 of *Lecture Notes in Computer Science*, pages 328–341. Springer Berlin Heidelberg, 2011.

[37] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *IMC*, pages 181–194, 2011.

[38] P. Hsiao, H. Kung, and K. Tan. Video over TCP with receiver-based delay control. In *NOSSDAV*, NOSSDAV, pages 199–208, 2001.

[39] S. Iyer, R. Kompella, and N. McKeown. Designing packet buffers for router linecards. *Transactions on Networking*, 16:705–717, 2008.

[40] R. Jain. A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks. *SIGCOMM*, 19(5):56–71, Oct. 1989.

[41] R. Jain. *Design and implementation of split tcp in the linux kernel*. PhD thesis, 2007.

[42] C. Jin, D. Wei, and S. Low. FAST TCP: motivation, architecture, algorithms, performance. In *INFOCOM*, volume 4, pages 2490–2501, Mar. 2004.

[43] W. Jingyuan, J. Wen, J. Zhang, and Y. Han. TCP-FIT: An Improved TCP Congestion Control Algorithm and its Performance. In *INFOCOM*, pages 1–12, Apr. 2011.

[44] D. Johnson. Performance Analysis of Mesh Networks in Indoor and Outdoor Wireless Networks. Master's thesis, University Of Pretoria, 2007.

[45] D. Johnson. *Re-architecting Internet Access and Wireless Networks for Developing Regions*. PhD thesis, University of California Santa Barbara, Goleta, Ca United States, 2013.

[46] H. Jung, S. Kim, H. Yeom, S. Kang, and L. Libman. Adaptive delay-based congestion control for high bandwidth-delay product networks. In *INFOCOM*, pages 2885–2893, Apr. 2011.

[47] C. Kellett, R. Shorten, and D. Leith. Sizing Internet Router Buffers, Active Queue Management, and the Lur'e Problem. *IEEE Decision and Control*, 2006.

[48] W. Kim and B. Lee. FRED fair random early detection algorithm for TCP over ATM networks. *Electronics Letters*, 34(2):152–154, Jan. 1998.

[49] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340, Mar. 2006.

[50] E. Kohler, M. Handley, and S. Floyd. Designing DCCP: congestion control without reliability. *SIGCOMM*, 36(4):27–38, Aug. 2006.

[51] M. Kuhlewind, S. Neuner, and B. Trammell. On the State of ECN and TCP Options on the Internet. In *Passive and Active Measurement*, pages 135–144, 2013.

[52] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. In *SIGCOMM*, pages 123–134, 2001.

[53] R. Kuschnig, I. Kofler, and H. Hellwagner. An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H.264/SVC. In *MMSYS*, pages 157–168, 2010.

[54] R. Kuschnig, I. Kofler, and H. Hellwagner. Improving Internet Video Streaming Performance by Parallel TCP-Based Request-Response Streams. In *Consumer Communications and Networking Conference (CCNC)*, pages 1–5, 2010.

[55] R. Kuschnig, I. Kofler, and H. Hellwagner. Evaluation of HTTP-based request-response streams for internet video streaming. In *MMSYS*, pages 245–256, 2011.

[56] S. Liu, T. Bacsar, and R. Srikant. TCP Illinois: a loss and delay-based congestion control algorithm for high-speed networks. In *Performance Evaluation Methodolgies and Tools*, 2006.

[57] D. Lu, Y. Qiao, P. Dinda, and F. Bustamante. Modeling and Taming Parallel TCP on the Wide Area Network. *Parallel and Distributed Processing Symposium*, 1:68b, 2005.

[58] M. Luby, T. Stockhammer, and M. Watson. IPT} Systems, Standards and Architectures: Part II - Application Layer FEC In IPTV Services. *IEEE Communications*, 46(5):94–101, May 2008.

[59] m. van der Schaar, s. Krishnamachari, s. Choi, and x. Xu. Adaptive cross-layer protection strategies for robust scalable video transmission over 802.11 WLANs. *IEEE Journal on Communications*, 21(10):1752–1763, Dec. 2003.

[60] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *IMC*, IMC, pages 90–102, 2009.

[61] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Mobile Computing and Networking*, pages 287–297, 2001.

[62] S. Mascolo and F. Vacirca. Congestion control and sizing router buffers in the internet. In *Decision and Control, and the European Control Conference (CDC-ECC)*, volume 2005, pages 6750–6755, 2005.

[63] T. Nguyen and S. Cheung. Multimedia streaming using multiple TCP connections. In *Performance, Computing, and Communications Conference (IPCCC)*, pages 215–223, Apr. 2005.

[64] K. Nichols and V. Jacobson. Controlling queue delay. *ACM Communications*, 55(7):42–50, July 2012.

[65] L. Ong and J. Yoakum. A Framework for Integrated Services Operation over Diffserv Networks. RFC 3286, May 2002.

[66] T. Ott, T. Lakshman, and L. Wong. SRED: stabilized RED. In *INFOCOM*, volume 3, pages 1346–1355, Mar. 1999.

[67] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. *SIGCOMM Commununications*, 28(4):303–314, Oct. 1998.

[68] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *Networking*, 8(2):133–145, Apr. 2000.

[69] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing*, pages 148–155, July 2013.

[70] D. Pei. Time for a TCP Benchmark Suite? 2008.

[71] V. Pejovic. *Adaptive and Resource-Efficient Rural Area Wireless Networks*. PhD thesis, University of California Santa Barbara, Goleta, Ca United States, 2013.

[72] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP, 2001.

[73] s. Molna, b. Sonkoly, and T. Trinh. A Comprehensive TCP Fairness Analysis in High Speed Networks. *Computer Communications*, 32(13-14):1460–1484, Aug. 2009.

[74] H. Schulzrinne, S. Casner, R. Fredrick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, Internet Engineering Task Force, July 2003.

[75] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Datagram Congestion Control Protocol (DCCP). RFC 6817, Mar. 2012.

[76] D. Stiliadis and A. Varma. Efficient fair queueing algorithms for packet-switched networks. *Networking*, 6(2):175–185, Apr. 1998.

[77] T. Stockhammer, G. Liebl, and M. Walter. Optimized H.264/AVC-based bit stream switching for mobile video streaming. *EURASIP*, 2006:1–19, Jan. 2006.

[78] I. Stoica, H. Zhang, and T. Ng. A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services. *Networking*, 8(2):185–199, Apr. 2000.

[79] J. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros. Network Coding Meets TCP: Theory and Implementation. *Proceedings of the IEEE*, 99(3):490–512, Mar. 2011.

[80] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *INFOCOM*, pages 1–12, Apr. 2006.

[81] A. Tang, L. Andrew, M. Chiang, and S. Low. Transport Layer. In *Encyclopedia of Computer Science and Engineering*, pages 2930–2938. Jan. 2009.

[82] A. Tang, J. Wang, S. Hegde, and S. Low. Equilibrium and Fairness of Networks Shared by TCP Reno and Vegas/FAST. *Telecommunication Systems*, 30(4):417–439, 2005.

[83] A. Tang, J. Wang, S. Low, and M. Chiang. Equilibrium of heterogeneous congestion control: existence and uniqueness. *IEEE/ACM Transactions on Networking*, 15(4):824–837, Aug. 2007.

[84] A. Tang, X. Wei, S. Low, and M. Chiang. Equilibrium of heterogeneous congestion control: optimality and stability. *IEEE/ACM Transactions on Networking*, 18(3):844–857, June 2010.

[85] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A Mechanism for Background Transfers. In *SIGOPS*, pages 329–343, 2002.

[86] C. Villamizar and C. Song. High Performance TCP in ANSNET. *SIGCOMM*, 24(5):45–60, Oct. 1994.

[87] A. Vishwanath, V. Sivaraman, and M. Thottan. Perspectives on router buffer sizing, 2009.

[88] F. Wang, M. Hamdi, and J. Muppala. Using parallel DRAM to scale router buffers. *IEEE Transactions on Parallel and Distributed Systems*, 20:710–724, 2009.

[89] D. Wei, P. Cao, and S. Low. Tcp pacing revisited. In *INFOCOM*. Citeseer, 2006.

[90] D. D. M. Welzl. MulTFRC: providing weighted fairness for multimediaapplications (and others too!). *SIGCOMM*, 39(3):5–12, June 2009.

[91] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS*, 36(SI):255–270, Dec. 2002.

[92] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Networked Systems Design and Implementation*, 2011.

[93] B. Wydrowski and M. Zukerman. GREEN: an active queue management algorithm for a self managed Internet. In *IEEE Communications*, volume 4, pages 2368–2372, 2002.